МОНИТОР

журнал для программистов — практиков

5.92

ISSN 0869—3889

sprite [sprait] n эльф, фея

Sprite

USER

Экспертиза деловых игр: проба пера

PROGRAMMER

Гипертекст: это вы можете!

Использование "мыши" в графических режимах

Некоторые предпочитают Zortech

Компьютерные забавы эльфов

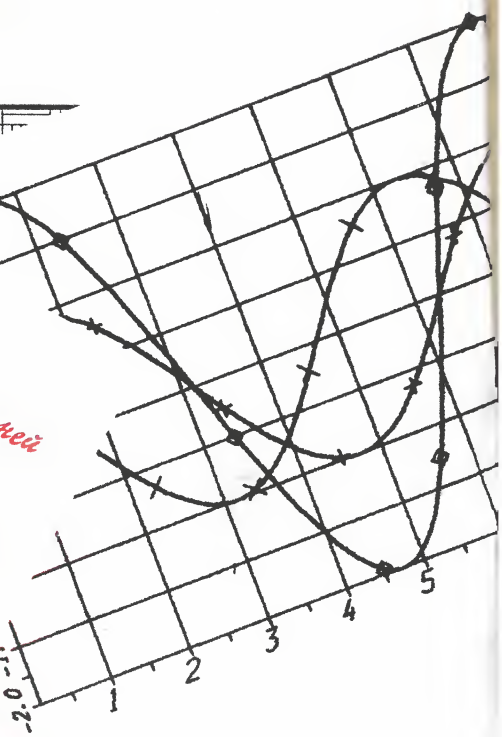
Почти по Ю.Семенову

Айсман, Штирлиц и Шульц, опрокинув по последней у старика Тюллера, собрались уходить.

— А Вас, Штирлиц, я попрошу остаться, — сдвинул брови Тюллер.

— Дружище! Вспоминая о Ваших связях на Востоке, прошу Вас, пощупите мне добротный графический пакет, русский желательно. Вашим, знаете, больше доверяю...

— Надо помочь старику, — подумал Штирлиц.



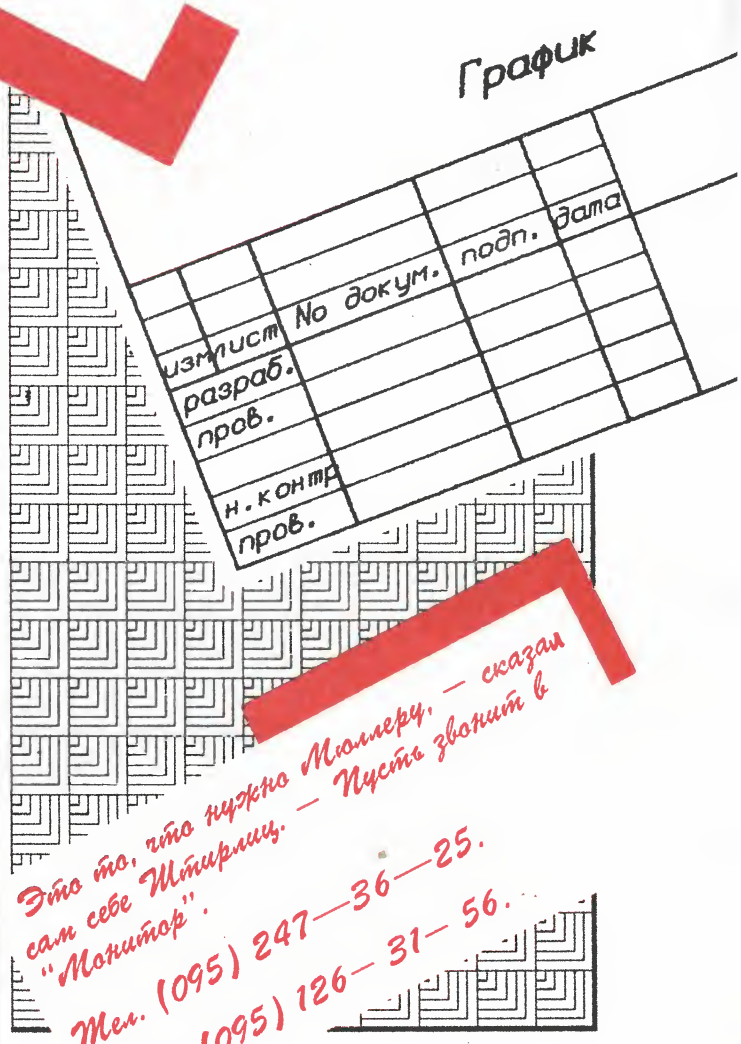
Информация к размышлению

ГРАФИЧЕСКИЙ ПАКЕТ ОБРАБОТКИ ДАННЫХ "АвтоГРАФ"

АвтоГРАФ предназначен для графического отображения массивов числовых данных и печати изображения с использованием 9 и 24-игольчатых принтеров . АвтоГРАФ функционирует в MS DOS 3.XX-5.XX на ПК типа IBM PC AT/XT с дисплеями EGA/VGA.

Достоинства пакета:

- представление графиков в равномерных, полулогарифмических и логарифмических осях;
- построение одновременно до 7 графиков;
- текстовый формат данных простой структуры;
- назначение угловых штампов и их заполнение (согласно требованиям ЕСКД);
- использование русских и латинских шрифтов для надписей под графиками и осями;
- настройка на распространенные типы принтеров;
- контекстно-ориентированный Help.



Это то, что нужно Тюллеру, — сказал сам себе Штирлиц. — Пусть звонит в "Монитор".

Тел. (095) 247—36—25.
Факс. (095) 126—31—56.

"очень своевременная книга!"

ОБШИРНЫЙ **КАТАЛОГ** ЗАРУБЕЖНЫХ ФИРМ

Электроника для Офисов, Банков и Торговли
Бытовая Электроника, Компьютеры и Периферия
Разнообразное Программное Обеспечение

Адреса, Телефоны
Предлагаемые Товары

Стоимость каталога - 985 рублей + *28%*
2 экземпляра - скидка 20 %

Бланки заказов и копии платежных поручений просьба высылать по адресу:

Москва, 117419, а/я 765, Каталог

Счет: НТО "Софт-Москва" р/с 467002 в КБ "Банк Развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г.Москва, МФО 201791

тел. (095) 247 3625

БЛАНК ЗАКАЗА

Название организации _____

Адрес для высылки Каталога _____

_____

Телефон _____Количество Экземпляров _____

МОНИТОР

Журнал для программистов-практиков

Редколлегия:

Д.Э.Бройтман
(главный редактор)

А.А.Винокуров
(отдел распространения)

В.В.Водолазский

И.Г.Гун

Н.А.Никитина
(отдел рекламы)

А.М.Миляков
(ответственный секретарь)

С.В.Макельский
(главный художник)

А.В.Орехов
(издательский директор)

А.В.Семенников

Л.И.Турок

Л.И.Устинова

В оформлении номера использованы рисунки

К.З.Гусалова
и А.С.Замулы

Подписано в печать с оригинал-макета 28.09.92.
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ.л.8.0.
Изд. N1971.
Цена договорная
Зак. 1600
Ф-ка "Картолитография"

Издатель:
Научно-техническое общество

Софт-Москва

Для писем: 117419, г.Москва,а/я 765,
тел. (095) 247-36-25,
факс 126-31-56

© "Монитор", 1992

Технологии программирования

Д.Гришечкин, В.Оленев, А.Тихонов — **Гипертекстовые системы** — **4**

В статье рассказывается о том, как средствами вычислительной техники ре зовать ассоциативное представление информации. Даются рекомендации по технологии создания гипертекстовых систем.

Б.Горбунов. — **Получение случайных чисел с нормальным распределением** — **9**

Описывается функция на языке Turbo Pascal, генерирующая случайные числа со стандартным нормальным распределением, имеющим нулевое среднее значение и единичную дисперсию.

Д.Солдатенков. — **Спрайты. Путь от графического редактора к компьютерным играм. Часть 1** — **10**

Демонстрируется технология создания и вывода на экран спрайтов - небольших объектов, постоянно меняющих свою форму и положение.

Базы данных

С.Крапивенко. — **Clipper или Paradox: попытка аргументированного выбора. Часть 2** — **14**

Продолжение разговора о преимуществах и недостатках Paradox и Clipper. Рассматриваются организация представления данных и техника работы с таблицами.

Л.Ливеровский. — **Что такое Clarion** — **19**

Статья рассказывает о возможностях интегрированной среды программирования Clarion.

Защита программ

И.Василенко, Е.Гулаков — **Защищайтесь, сударь! Часть 2** — **26**

Во второй части статьи читатели сначала познакомятся с набором средств, доступных программисту, которые он может использовать в целях создания защиты от копирования, а затем с программными реализациями на языке Turbo C некоторых методов нестандартного форматирования, рассмотренных в части 1 статьи, опубликованной в предыдущем номере.

Игры

Дороги, которые мы выбираем — **31**

В первой из серии статей-экспертиз игрового программного обеспечения, подготовленных научно-технической фирмой "Инфорком", — анализ лучшей игры-имитатора 1990 года "Railroad Tycoon" фирмы MICROPROSE.

Мир графики

А.Поликарпов. **EGA и VGA в текстовых режимах** 36

Статья посвящена программированию EGA и VGA адаптеров в текстовых режимах.

В.Водолазкий, В.Головин, Е.Короткова. **Паскаль + Ассемблер = ... или как использовать "мышь" в графических режимах** 41

Описывается технология написания процедур на Ассемолере для их использования в программах на Turbo Pascal. В качестве примера рассмотрен интерфейс, позволяющий использовать манипулятор "мышь" из программ на Turbo Pascal.

Технологии программирования

М.Черкашин. **Вычисляем по формуле ...** 51

Вычислить выражение может оказаться полезным. Чтобы написать эту процедуру просто, нужно знать некоторые технические приемы. Вторая статья из цикла статей, посвященных написанию компиляторов.

Сети

PC_Multi — инструмент для профессионалов 53

Использование PC_Multi позволяет организовать одновременную работу нескольких пользователей на одной ЭВМ, предоставив каждому свой терминал.

Языки программирования

А.Сидоров. **Некоторые предпочитают Zortech** 55

Статья рассказывает о возможностях профессионального компилятора Zortech C++.

САПР

Р.Глуховский. **Работа с DXF файлами в Автокаде 11** 58

Графический редактор Автокад является мощной системой автоматизированного проектирования с открытой архитектурой. Достоинством системы с открытой архитектурой является ее программируемость и адаптация к требованиям конечного пользователя. В этой статье рассматривается один из возможных способов программирования в Автокаде.

Бизнес

MERISEL: опыт цивилизованнои работы с дилерами 64

Гипертекстовые системы

Д. Гришечкин, В. Оленев, А. Тихонов

ТЫ купил HyperRef ?!

— Идея покушения проста, как колумбово яйцо.
— Как и то, что вас ждет Панкрац,— дополнил его вывод полицейский комиссар при допросе.

Я. Гашек

Идея гипертекста (ГТ), как и любая другая здравая идея, проста и состоит в том, чтобы дать возможность человеку воспринимать информацию в определяемой им самим последовательности. Целью настоящей статьи является введение читателя, работающего на ПЭВМ, совместимых с IBM PC, в проблематику создания гипертекстовых систем (ГТС). Соблазн разработки ГТ продуктов захватил многие коллективы и отдельных программистов. В нашей коллекции имеются более десятка демонстрационных версий отечественных гипертекстовых систем, собранных на выставках в 1991-92 годах. Авторы сами не избежали гипертекстового поветрия, разработав систему для работы с большими электронными документами под названием HyperRef, работоспособная демонстрационная версия которой может быть получена либо у авторов, либо в редакции.

Итак, идея !

ГТС представляет собой реализацию средствами вычислительной техники ассоциативного представления информации. Ассоциативность свойственна человеку. Каждый из нас знает, как трудно бывает изложить на бумаге сложившиеся в голове мысли, как много мыслей при этом теряется! Здесь проявляются издержки нашей фонетической письменности (в отличие, например, от китайской и японской), которая поддерживает линейную организацию информации. Естественно имеются и счастливые исключения, такие как герой М.Е. Салтыкова-Щедрина, который, прочитав некий документ, заявил: " Понять не могу, но ответить смогу".

Существует большое число определений ГТС, наиболее характерные из которых приводятся на врезке. Для нас существенно, что гипертекст представляет собой множество фрагментов статической информации, объединенных в сеть. В качестве информационных фрагментов могут выступать текст, графический образ и даже программа.

В текстовых и графических фрагментах могут быть некоторым образом (цветом, шрифтом, рамкой и т.д.) выделены специальные поля, используемые для связывания информационных фрагментов в гипертекстовую сеть. Такие поля в дальнейшем мы будем именовать кнопками. В гипертекстовой сети возможно выделение нескольких подсетей, связанных между собой.

Активизация связи приводит к переходу пользователя к следующему информационному фрагменту. В зависимости от типа связи переход может быть постоянным или временным. В последнем случае пользователь получает возможность ознакомиться с примечанием, пояснением, толкованием термина и вернуться к первоначальному информационному фрагменту. Процесс "путешествия" пользователя по информационным фрагментам называется навигацией. ГТС позволяют осуществлять последовательный просмотр информации, как это мы делаем, листая книгу, иерархический просмотр (глава, параграф, страница) и самое важное — произвольную навигацию по ГТ сети, определяемую личным опытом, интересами и не в последнюю очередь настроением пользователя. Последнее накладывает дополнительную ответственность на автора ГТ документа, который должен не только обеспечить высокое качество содержания информационных фрагментов и осмысленные их взаимосвязи, но и предвидеть возможные маршруты прохождения по нему пользователя. В качестве автора ГТ документа может выступать и сам пользователь.

Объекты гипертекстовой сети образуют следующую иерархию: сеть, подсеть, фрагмент, кнопка. Соответственно в ГТ могут присутствовать связи типа: кнопка-кнопка, фрагмент-кнопка, кнопка-фрагмент, кнопка-подсеть, фрагмент-подсеть, подсеть-фрагмент, подсеть-кнопка.

Обычно прикладные ГТС поддерживают один или несколько

типов связей. Например, если условиться, что текстовый фрагмент представляет собой полностью видимую на экране страницу, то вполне достаточно наличия связей кнопка-страница и страница-страница. Иллюстрация такой ГТ сети приведена на рис.1. Если же ГТС допускает наличие фрагментов, размер которых превышает окно, в котором они демонстрируются, то необходимыми оказываются связи типа кнопка-кнопка и фрагмент-кнопка так, чтобы при переходе по связи осуществлялось бы правильное позиционирование фрагмента в окне (рис.2). Аналогично вводятся и связи кнопка-подсеть, фрагмент-подсеть (рис.3).

Наличие связей различных типов ставит вопрос о способе использования их для навигации. Действительно для активизации связи кнопка-фрагмент или кнопка-кнопка пользователю достаточно "нажать" требуемую кнопку маркером (с помощью клавиатуры или мышки). Несколько сложнее с активизацией связей типа фрагмент-кнопка и фрагмент-фрагмент, фрагмент-подсеть. Самым простым способом является резервирование специальной "горячей клавиши" для вызова меню, в котором пользователю предъявляются имена всех фрагментов (кнопок), с которыми связан данный фрагмент. Переход осуществляется после выбора целевого объекта в меню.

Существенной особенностью гипертекстовых систем является их ориентация на активную работу пользователя. Осуществляя навигацию по информационным фрагментам, пользователь получает возможность по разному представлять информацию, что в конечном счете облегчает ее понимание и обеспечивает создание новой информации. Все это позволяет говорить о ГТС, как о знаниеориентированных системах.

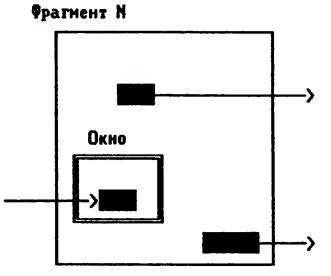


Рис. 2. Связь фрагмент–кнопка

Зачем нужны ГТ системы?

Информационные системы. Прежде всего ГТ прекрасно приспособлен для создания информационных систем особенно в плохо структурированных предметных областях, а также когда пользователь не может четко сформулировать свои информационные потребности, а делает это в процессе поиска информации. Традиционные методы информационного поиска, например, по ключевым словам, могут являться дополнительным средством в прикладных ГТС, обеспечивая эффективный доступ к большим информационным массивам.

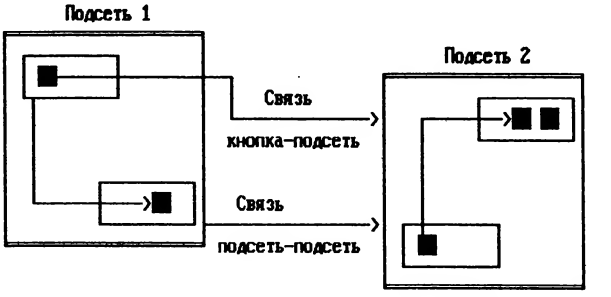


Рис. 3. Гипертекстовая сеть, содержащая подсети

Обучающие системы. Ассоциативные связи между информационными фрагментами облегчают усвоение знаний и делают ГТ незаменимым для создания средств обучения. Гибкость ГТ позволяет преподавателю индивидуализировать процесс обучения, а также сделать его наглядным и интересным при умеренных затратах на разработку обучающих курсов.

Системы интерактивной помощи. Каждый программист, хотя бы раз работавший с компиляторами фирмы Борланд, оценил возможность получения интерактивных справок, особенно по библиотекам системы программирования. Достаточно подвести курсор к ключевому слову или имени функции и нажать Ctrl-F1, как появится ее описание, сопровождаемое примерами использования. В систему Турбо Паскаль для Windows включены мощные средства для создания собственных интерактивных гипертекстовых справочных систем с использованием различных фонтов и графики. В качестве другого примера такой системы можно привести средства, поставляемые на уровне исходных текстов и включенные в пакет Object Professional фирмы TurboPower Software.

Системы сопровождения программных систем. Естественной экологической нишей для ГТ является его применение при разработке больших программных систем. Тексты программ и комментарии могут быть помещены в узлы гипертекстовой сети, а связи использоваться для автоматической генерации документации, ведения версий разработки, управления проектом, поиска ошибок. Данное направление использования ГТ по крайней мере для ПЭВМ, совместимых с IBM PC, в настоящее время открыто, и появление простых, дешевых и удобных систем поддержки в значительной степени увеличит производительность работы программистов, а главное облегчит и уменьшит стоимость процесса сопро-

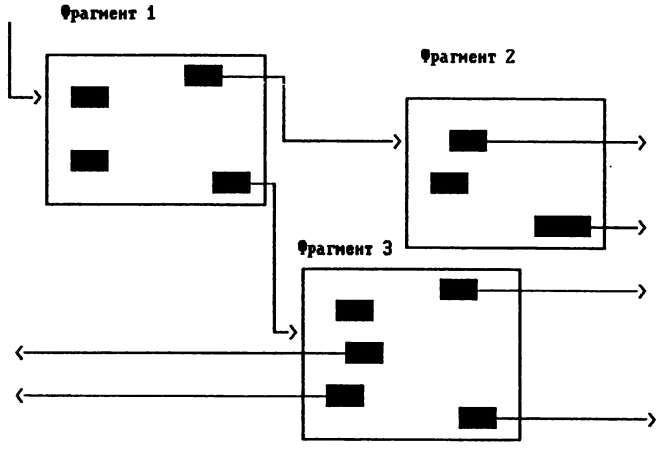


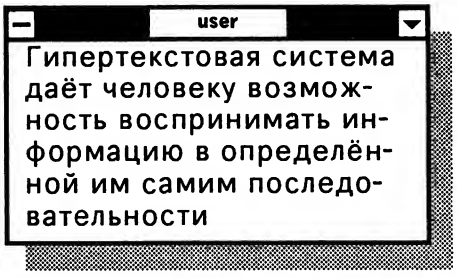
Рис. 1. Гипертекстовая сеть информационных фрагментов

вождения.

Процессоры идей. Данный тип ГТС используется для организации процесса создания и сопровождения больших документов, содержащих текст, графику, информацию из баз данных и электронных таблиц. Конечной целью является генерация линейного и удобочитаемого документа (в виде книги, доклада, законопроекта, отчета, например, о деятельности фирмы за квартал и т.д) по разнородным и разноплановым данным. Здесь существенным является отслеживание влияний изменений данных в отдельных информационных фрагментах на документ в целом и его корректная реорганизация для различных приложений (например, рекламного буклета, проспекта о выпуске акций, доклада на собрании акционеров). Кроме этого немаловажным является поддержка одновременной работы нескольких авторов, их взаимодействия в локальной сети или с помощью электронной почты, распределенного хранения информации, ведения версий документов, согласования их окончательного содержания.

Мультимедиа. Последние годы термин ГТ постепенно заменяется на гипермедиа и мультимедиа (ММ), при этом в дополнение к тексту, графике и программам в качестве информационных фрагментов могут служить звуковое сопровождение и динамическая графика. В статье мы намеренно не касаемся этих вопросов и отсылаем читателя к прекрасному обзору [1] в основном из-за отсутствия соответствующих технических средств у среднего российского пользователя ПЭВМ.

В настоящее время идет интенсивная разработка технических, программных средств для создания ММ продуктов, а также стандартов представления информации. Так, фирма IBM выпустила специальную систему для работы с ММ на основе PS/2 57 SLC стоимостью около 6 000 долларов, а MicroSoft - расширение Windows 3.1 для тех же целей. По свидетельству обозревателя журнала Byte [2] ММ в течение следующего десятилетия изменит внешний вид и технологию создания и работы с программными продуктами в гораздо большей степени, чем это сделали графические пользовательские интерфейсы за последние пять лет.



Простая ГТС

Посмотрим теперь, что же нужно для создания собственной ГТС. Сравнительно немного. Прежде всего это база данных, где будут храниться текстовые (графические) фрагменты, а также связи между ними. Не обойтись и без средств демонстрации пользователю информационных фрагментов и связей, позволяющих "путешествовать" по ГТС, и наконец, без средств конструирования, обеспечивающих ввод и редактирование информационных фрагментов, создание связей между ними.

Если Вы работаете с Турбо Паскалем версии 6, то для создания небольших ГТС целесообразно использовать аппарат так называемых коллекций и ресурсов. Коллекция представляет собой объектно-ориентированное расширение массива, позволяющее легко осуществлять манипулирование переменным числом объектов различной структуры. Ресурсы позволяют хранить объекты, например, текстовые фрагменты, на устройствах прямого доступа, используя для поиска их имена.

Широко распространенным подходом, применяемым для создания гипертекстовых баз данных, является аппарат так называемых сбалансированных двоичных деревьев (B-деревьев). Каждому ГТ объекту сопоставляются имя фиксированной длины и адрес начала объекта на диске. Средства работы с B-деревьями позволяют упорядочить имена объектов и иметь к ним быстрый доступ. Все обращения к ГТ объектам осуществляются по имени, в B-дереве отыскивается адрес объекта, после чего осуществляется его выборка с устройства прямого доступа. При использовании B-деревьев в связи, указывающей на ГТ объект, хранится его имя. В принципе можно обойтись и без B-деревьев, храня в связях непосредственно адреса объектов, однако при этом создаются серьезные неудобства в процессе организации связей между ГТ объектами и построением перечней (подмножеств) объектов, так как приходится читать сами объекты для определения их имен.

Исходные тексты программ, необходимых для построения ГТС и в частности для работы с B-деревьями, приводятся в 11-м номере журнала Dr. Dobb's Journal за 1990 год, который практически полностью посвящен гипертексту.

Еще одной проблемой, возникающей при реализации ГТС является построение пользовательских интерфейсов. Сравнительно быстро интерфейсы с пользователем (с использованием как клавиатуры, так и мыши) в среде ДОС могут быть разработаны с помощью Turbo Vision, а в среде Windows - с помощью Object Windows. Эти средства поставляются в составе систем программирования Турбо Паскаль версии 6 и компиляторов фирмы Borland, работающих в среде Windows. Компактные и красивые средства для построения графических пользовательских интерфейсов в среде ДОС предоставляет пакет Tegl Windows Toolkit II фирмы Tegl System Corporation. Версия 1.10 предназначена для работы с Турбо Си 2.0 и Турбо Паскалем 5.5, но доработки, сделанные Е.Б. Киселем, позволяют работать и с Паскалем 6.0. Если в Вашей ГТС присутствует графика в формате PC PaintBrush, то для демонстрации картинок могут быть использованы программные средства, исходные тексты которых приведены в [3].

Наконец, для заполнения гипертекстовой базы данных информационными фрагментами и установления между ними связей необходимы специальные средства, реализуемые либо в виде транслятора, либо в виде интерактивной программы. Каждый из этих подходов имеет свои преимущества и недостатки. Транслятор проще в реализации, однако менее нагляден, и с его помощью труднее создавать и сопровождать большие ГТ документы. С другой стороны, ин-

терактивный конструктор для гипертекста более сложен для реализации, так как требует текстового редактора, средств для разметки полей и визуализации гипертекстовой сети. В большинстве простых гипертекстовых систем (по крайней мере отечественных) ввод осуществляется с помощью языка разметки текста. В процессе создания гипертекстовых баз данных с помощью HyperRef мы используем утилиту для первоначального ввода информационных фрагментов, организацию связей, редактирование и модификацию гораздо удобнее осуществлять с помощью диалогового конструктора.

Заметим, что практически все примитивы для создания средств работы с ГТС присутствуют в системе программирования PDC Prolog 3.21 датской фирмы Prolog Development Center (PDC). Вторая версия этой системы распространялась фирмой Borland под именем Турбо Пролог. Начиная с 1992 года, в нашей стране ее распространением занимается информационно-программный центр Интеллект-Банк в Санкт-Петербурге. Эта система обладает рядом интересных особенностей, что может служить предметом отдельного разговора. Для нас существенно, что в нее встроены мощные и удобные средства для работы с базами данных, а также средства конструирования пользовательских интерфейсов. Не скроем, что для того, чтобы писать на Прологе, программисту, привыкшему к любому процедурному языку, приходится проходить психологическую ломку. Однако после освоения системы производительность труда повышается минимум в два-три раза. Все это и обусловило наше решение писать свою ГТС на Прологе. Оказалось, что мы не одиноки, и фирма PDC сама выпустила набор инструментальных средств для создания ГТС под названием PDC Prolog HyperText ToolBox, стоящий ни много ни мало, а 330 долларов.

Большинство ГТС написано на Си или Паскале. Однако существуют попытки реализации ГТС средствами стандартной СУБД. Наиболее известная из них - ГиперЛог [4], созданная в виде надстройки к СУБД Revelation (Апокалипсис). Однако использование СУБД приводит, по крайней мере для ПЭВМ, к большим накладным расходам как при работе с информационными фрагментами, так и при построении пользовательских интерфейсов. Если у Вас все-таки возникнет непреодолимое желание реализовать ГТС на Клиппере, подумайте дважды!

... А из зала мне кричат: "Давай подробности!"

Выше мы рассмотрели базовые средства, необходимые для работы практически любой ГТС Однако в программировании, как и в искусстве, важны подробности, а реализация любой программной системы связана с решением множества больших и маленьких проблем. Основной проблемой является выбор экологической ниши разработки, а также сбалансированного набора выполняемых функций и изобразительных средств. Дело в том, что наметилось явное перепроизводство простых ГТС реализующих джентльменский набор базовых функций, при отсутствии в среде ДОС удовлетворительных разработок для поддержки разработки больших программных проектов и процессоров идей. Мы далеки от того, чтобы давать рекомендации на все случаи жизни, но рассмотрим несколько достаточно простых средств, реализация которых может украсить ГТС.

Прежде всего это приписывание свойств связям между информационными фрагментами. Оказалось, что наделение связи силой и "цветом" значительно повышает потребительские свойства ГТС.

Сила связи используется для динамического формирования подсетей. Серьезной проблемой для ГТС является информационная перегрузка пользователя, то есть возможность "заблудиться" в процессе навигации по сети. Учет сил связей между объектами не позволяет пользователю удалиться слишком далеко, а задание пороговой силы обеспечивает плавное регулирование "радиуса действи " пользователя.

"Расцвечивание" связей также позволяет выделить отдельные подсети, например, в материаловедческом справочнике подсеть, описывающую технологию изготовления некоторого материала. Приписывание связям таких свойств как направленность и транзитивность обеспечивает автоматическое построение различных пользовательских ракурсов гипертекстовой сети, например, перечней и оглавлений. В конечном счете механизм "расцвечивания" связей позволяет реализовать такой мощный аппарат представления знаний, как семантические сети. Оказалось, что построение средств для работы с семантическими сетями на Прологе естественно, лаконично и не требует существенных затрат.

При создании больших баз данных возникают проблемы с хранением информации. Желательно уже при разработке ГТС спроектировать средства сжатия информации. Мы вспомнили об этом только тогда, когда объем базы данных по одной из разработок превысил 3 Мбайта. Своеобразие вопроса состоит в том, что средства сжатия информации должны обеспечивать высокую скорость по крайней мере распаковки, достаточную степень сжатия (не менее двух) даже небольших фрагментов информации. На наш взгляд, указанным требованиям удовлетворяет LZW-алгоритм [5] с предварительным кодированием повторяющихся последовательностей, а для хранения графики целесообразно использовать формат GIF фирмы CompuServe [6], обеспечивающий существенно большее сжатие по сравнению с PCX.

Вспомним теперь о том, ради кого все это затевалось — о пользователе. От ГТС и пользовательских интерфейсов он ждет интуитивности поведения, прозрачности и удобства. Все команды, связанные с навигацией по гипертекстовой сети ("перейти", "вернуться", "смотреть", "листать"), должны выдаваться по крайней мере тремя способами: с помощью меню, "горячих клавиш" и мышки. Пользователь должен иметь возможность вернуться в информационные фрагменты, где он уже побывал, пометить интересующий его фрагмент "книжной закладкой", вывести фрагмент на принтер или в файл, иметь возможность запомнить состояние системы с тем, чтобы в следующем сеансе восстановить состояние системы и продолжить работу с того же места. Сюда же относятся средства, называемые браузерами и позволяющие увидеть структуру гипертекстовой сети с различной степенью подробности.

Желательно предоставить пользователю и регулярные средства поиска информации в ГТ сети. При этом каждому гипертекстовому объекту приписывается набор атрибутов. Атрибут может иметь числовое или строковое значение. В качестве атрибутов могут служить, например, свойства материалов, стоимость, имя автора данного фрагмента, время его последней модификации и т.д. Существенно, что как набор атрибутов, так и их значения должны легко модифицироваться в процессе разработки ГТ сети. Для кодирования и быстрого поиска атрибутов и значений также удобно использовать B-деревья. На них же естественно реализуются и сложные запросы, объединенные логическими связками. Опять-таки Пролог позволяет встраивать эти средства в готовую ГТС с минимальными накладными расходами.

Основная трудоемкость при создании ГТ сетей падает на организацию и верификацию связей. Поэтому Вы завоюете симпатии своих пользователей, если облегчите им жизнь, реализовав средства полуавтоматического формирования связей. Сделать это можно следующим образом. На первом проходе по вводимым в сеть текстам во взаимодействии с пользователем строятся словари. Дело в том, что русский - флексивный язык с изменяемыми основами слов, с большим количеством приставок и суффиксов и автоматическое построение словарей вряд ли целесообразно. Далее пользователь конструирует связи между словарными статьями. Построение связей может проводиться как в явном виде, так и индуктивно путем автоматического анализа уже имеющейся ГТ сети.

Наконец, в процессе установления связей при появлении словоформы, имеющейся в словаре, пользователю предъявляется список связей и целевых объектов, с которыми можно установить связь. Установка связей осуществляется выбором объектов в соответствующих меню.

Гипертекст, как и электрон в известной цитате, неисчерпаем, поэтому закончим на этом, предоставив читателю возможность продумать, как организовать работу с различными версиями информационных фрагментов и связей, хранящихся в ГТ сети, как поддерживать актуальность динамически импортируемых фрагментов из других программных систем (баз данных, электронных таблиц) и о многом другом.

Определения термина гипертекст

Гипертекст имитирует способность человеческого интеллекта осуществлять хранение больших объемов информации и поиск в них посредством ассоциаций в процессах коммуникации и мышления.
(В.Буш)

Гипертекст — средство, помогающее людям писать, думать, показывать.
(Т.Нельсон)

Гипертекст — это способ хранения и манипулирования информацией, при котором она хранится в виде сети связанных между собой узлов.
(М. Форрестер)

Гипертекст — технология работы с текстовыми данными, позволяющая устанавливать ассоциативные связи – "гиперсвязи" – между фрагментами, статьями в текстовых массивах и благодаря этому допускающая не только последовательную, линейную работу с текстом, как при обычном чтении, но и произвольный доступ, ассоциативный просмотр в соответствии с установленной структурой связей. Текст как бы получает дополнительные измерения будучи одновременно организованным и от начала к концу, и по тематическим линиям, по индексам, библиографическим указателям.
(С.Новосельцев).

Гипертекст — это форма организации текстового материала, при которой его единицы представлены не в линейной последовательности, а как система явно указанных возможных переходов, связей между ними.
(М.Субботин)

Гипертекст — в американских публикациях гипертекстом называют как саму форму организации текстового материала (нелинейную, сетевую), так и технологию, без которой невозможна такая организация материала в широких масштабах.
(М.Субботин).

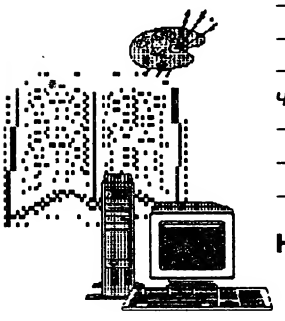
Гипертекстовая технология — средство охвата различных аспектов проблемы, идеи и подходов к ее решению, аргументов и фактов, накапливаемых и представляемых в их взаимосвязи, независимо от какой–либо априорной, заранее принятой позиции и исследуемой в их совокупности.
(М.Субботин).

Литература

1. Новосельцев С. Мультимедиа - синтез трех стихий. // Компьютер Пресс, ч. 1, 1991, №7; ч. 2, 1991, №8; ч. 3, 1991, №11.
2. Yager T. Information's Human Dimension. // Byte, №12, 1991.
3. Тихонов А.И. Показать картинку... // Монитор, №3, 1991.
4. Зефирова В.Л., Челнокова В.М. ГИПЕРЛОГ: Гипертекстовая система с логико-смысловой навигацией. // Научно-техническая информация. Сер. 2. Информационные процессы и системы, 1990, №4, с.10-14.
5. Bolton D. Writing your own PKZIP Utility. // Program Now, Febr, 1991.
6. Rimmer S. Bit-Mapped Graphics. // WindCrest, Blue Ridge Summit, Pa., 1991.

Необходимо создать электронную энциклопедию, справочник, книгу?

Гипертекстовая система **HyperRef** *поможет Вам!*



Мы предлагаем:

– интеграцию в электронном документе текста, графики и программ;
– охрану авторских прав создателей документа;
– объем документов ограниченный только емкостью диска;
– мощный интерактивный конструктор гипертекста;
– настройку на индивидуальные информационные потребности читателя;
– встроенные средства контроля семантической близости понятий;
– "книжные закладки" и печать содержимого документа;
– гипертекстовый справочник–руководство по HyperRef.

Немного усилий, и Ваше произведение найдет своих читателей!

Телефон: (095) 247–36–25
Факс: (095) 126–31–56

Получение случайных чисел с нормальным распределением

Те, кому необходимо иметь качественный генератор случайных чисел с нормальным распределением, наверняка сталкиваются с трудностями поиска подходящего алгоритма. Автору пришлось пересмотреть большое количество специальной литературы, прежде чем он нашел такой алгоритм в [1]. На базе него на языке Turbo Pascal была написана функция, выдающая случайные числа со стандартным нормальным распределением, имеющим нулевое среднее значение и единичную дисперсию (см. листинг).

Для образования пары независимых случайных чисел с нормальным распределением используется пара случайных чисел с равномерным распределением в интервале от 0 до 1. Поскольку используемый метод образует сразу два числа, то функция имеет "двухтактный" режим работы: при первом обращении к ней образуется пара чисел и одно из них передается в качестве значения функции, а при втором обращении осуществляется только передача второго числа. При начальном обращении к функции из программы предусмотрена инициализация генератора случайных чисел.

Последовательность чисел, получаемая с помощью функции, является псевдослучайной, т.е. имеет некий период повторения. При попытке определить этот период экспериментально было установлено, что он превышает по крайней мере 100 млн. значений.

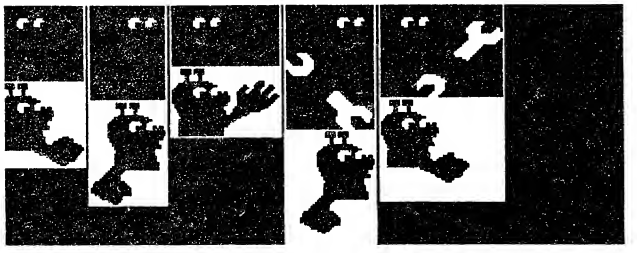
Б. Горбунов

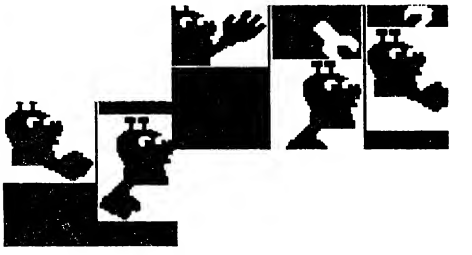
Листинг.

```
{  Пример. Функция - генератор случайных чисел   }
{  со стандартным нормальным распределением   }
function NormRandom : double;

const
    Ready       : boolean = false;
    SecondValue : double = 0.0;
    Start       : boolean = true;

var
    Argument  : double;
    Factor    :.double;

begin
    If (Start = true) then
    begin
        Start := false;
        Randomize;
    end;
    If (Ready = false) then
    begin
        Ready := true;
        Factor := Sqrt(-2.0 * Ln(Random));
        Argument := 2.0 * Pi * Random;
        SecondValue := Factor * Sin(Argument);
        NormRandom := Factor * Cos(Argument);
    end
    else
    begin
        Ready := false;
        NormRandom := SecondValue;
    end;
end; { function NormRandom }
```

ЛИТЕРАТУРА

1. Справочник по специальным функциям с формулами, графиками и математическими таблицами. Под ред. М.Абрамовица и И.Стиган.: Пер.с англ.под ред.В.А.Диткина и Л.Н.Кармазиной. М.: Наука, Гл.ред.физ.-мат.лит., 1979, с.746.





Спрайты. Путь от графического редактора к компьютерным играм

Часть 1

Д. Солдатенков

Всем известна магическая притягательная сила, которой обладают компьютерные игры. Часто не только дети, но и взрослые дяди и тети часами с азартом сражаются с вычислительной машиной. Иногда новую компьютерную игру осваивают, не прерываясь даже на ночь.

Успех компьютерной игры на 90% определяется ее внешним видом, который формируется художником при помощи графического редактора. В задачу же программиста входит разработка технологической цепочки, которая позволит перенести созданные художником образы на игровой экран. В каждой игре принимают участие небольшие объекты, постоянно меняющие свою форму и положение. Почти всегда это объекты не прямоугольной формы, имеющие ручки, ножки или в лучшем случае стабилизаторы. Но удобнее всего работать с прямоугольниками, поэтому непрямоугольник вписывают в прямоугольник и задают соответствующие области прозрачности. Полученный таким образом объект назвали "спрайт" (видимо за сходство с этими сказочными существами в способности к бесшумному перемещению, перелету).

Спрайты — основной способ вывода на экран в компьютерных играх, но очень часто они встречаются и в "серьезных" программах. Например, курсор манипулятора типа "мышь" в графическом режиме — спрайт. Применение спрайтов может сделать любую программу гораздо привлекательнее и эффектнее. Приводимая здесь технология изготовления и вывода спрайтов успешно используется уже на протяжении ряда лет в лаборатории инструментальных средств Московского Института Новых Технологий Образования и вызывает большой интерес у программистов. Допустим, что спрайт нарисован. Как ввести его в игру?

В технологической цепочке, связывающей графический редактор с игровым экраном, можно выделить два этапа. Первый этап — перевод данных из формата графического редактора в формат, с которым работает программа, а второй — вывод на экран. Рассмотрим сначала второй этап, поскольку именно способ вывода определяет формат данных для него. Как правило, на процедуры вывода спрайтов накладываются жесткие требования по скорости выполнения, поэтому приходится работать непосредственно с видеопамятью. Особый интерес при этом представляет, конечно, отработка областей прозрачности. Данные для задания области прозрачности можно хранить в виде отдельной маски или считать прозрачным один из цветов. В первом случае данные занимают больше места, во втором — не полностью используются цветовые возможности экрана. Способы отработки прозрачности сильно зависят от режима, в котором находится экран. Наиболее простым для вывода спрайтов является 256-цветный VGA режим. В этом режиме каждый байт кодирует свой пиксел и программист может быть полностью избавлен от работы с отдельными битами. Немного сложнее
4 - цветный CGA режим. Здесь каждый пиксел кодируется двумя битами. Можно выводить изображение спрайта двумя последовательными операциями — И и ИЛИ, при этом области прозрачности в первом случае заполняются единицами, а во втором — нулями. Но наиболее распространенным и наиболее сложным является, конечно, 16-цветный EGA режим. В этом режиме видеопамять состоит из четырех плоскостей, состояние соответствующего бита каждой плоскости кодирует один пиксел экрана. Программисту предлагается богатый выбор. Можно использовать маскирование битов и плоскостей, логические операции, вращение, писать байты вдоль (режим записи 0) и поперек (режим записи 2). Мы рассмотрим подробно только один из возможных способов вывода спрайтов. Он реализован в программе EGASprite, написанной на языке Турбо Паскаль версии 6.0 с использованием встроенного ассемблера. Во время работы этой программы по экрану прыгает желтый человечек. При использовании данного способа вывода, маска для областей прозрачности задается отдельно и спрайт может содержать все 16 цветов. Запись происходит в режиме 0, при этом данные, приходящие с процессора, попадают во все открытые плоскости и во все открытые биты через регистр задвижки, предварительно вращаясь. Каждый приходящий байт влияет на 8 точек экрана, лежащих рядом в одной строке, причем самый старший бит соответствует самой левой

Анонс

"Монитор" — версия 6.92

Д. Солдатенков. Спрайты. Путь от графического редактора к компьютерным играм (часть 2)

Демонстрируется технология создания и вывода на экран спрайтов - небольших объектов, постоянно меняющих свою форму и положение.

В. Водолазкий.

Коммерческие системы шифрования: основные алгоритмы и их реализация

Статья посвящена алгоритму шифрования коммерческой информации DES. Описан принцип реализации систем с открытым ключом. Впервые в отечественной литературе приведено описание систем идентификации абонента по так называемой "электронной подписи".

О.Сурин. Мирное применение вирусной технологии

Если Вы умеете создавать вирусы, то Ваши знания могут быть направлены в полезное русло. Защита программ от запуска с помощью вируса - тема предлагаемой статьи.

Постижение истории Экспертный анализ популярной игровой программы Civilization фирмы Microprose.

С. Крапивенко. Clipper или Paradox: попытка аргументированного выбора (часть 3)

Третья часть статьи, сравнивающей возможности Clipper и Paradox.

В. Рабец. Работа с DOS и BIOS в обход прерываний

Статья рассказывает о методах нахождения точки входа в обработчики прерываний DOS и BIOS и обращения непосредственно к этим обработчикам. Приведен текст программы, реализующей данные методы на Turbo Pascal 6.0.

А. Ермолов. Что такое IPX и как его использовать в своих целях (часть 1)

В статье приводится краткое описание программы симметричного обмена информацией - IPX, входящей в состав сетевых операционных систем семейства NetWare фирмы Novell. Приводятся простейшие примеры использования программного интерфейса IPX в пользовательских программах.

С. Курковский. Алгоритмы сортировки

Достоинства и недостатки пяти различных методов сортировки.

programmer

Спрайты — основной способ вывода на экран в компьютерных играх, но очень часто они встречаются и в серьёзных программах

точке. Мы будем хранить для каждой плоскости свой байт данных и писать каждый раз только в одну плоскость. Получается, что для того чтобы заполнить 8 точек спрайта на экране, нам потребуются 5 байтов данных: 1 — для прозрачности и 4 — данные для 4 плоскостей. Эти 5 байтов будем называть "байты видеоданных (VD)". Если смещение от начала экрана до первого пиксела спрайта не равно 8, то данные из одного байта VD должны попадать в разные байты видеопамяти (рис.1).

В этом случае будем использовать регистр вращения, который вращает данные, поступающие с процессора, перед помещением их в регистр задвижки. Каждый байт данных будет писаться в два приема. Сначала вращаем его вправо и пишем в левый байт видеопамяти, маскируя левые биты (рис.2).

Затем вращаем влево и пишем в правый байт, маскируя правые биты (рис.3).

Для того чтобы перемещать спрайт, надо быстро стирать его и рисовать в новом месте. Если не используются манипуляции со страницами видеопамяти, то для того чтобы не было заметно миганий изображения, программа должна перерисовывать спрайт за время обратного хода луча. Данный алгоритм позволяет сделать это даже с использованием стандартной графики языка Турбо Паскаль при условии, что тактовая частота процессора не ниже 10 МГц. Если быстродействие процессора невелико, то для успешной работы программы придется уменьшить размер спрайта. В программе (листинг) использован один необычный прием. Для того чтобы задать области прозрачности, необходимо задать битовую маску, но средствами языка Паскаль нельзя получить удобный доступ к отдельным битам данных. Зато это можно сделать, вызвав встроенный ассемблер. Создадим процедуру, состоящую только из данных, описанных на языке ассемблера в двоичном формате, и будем обращаться с указателем на эту процедуру просто, как с указателем на массив данных. Таким образом, можно получить графический редактор прямо в среде языка Турбо Паскаль. Так как перечеркнутые нули

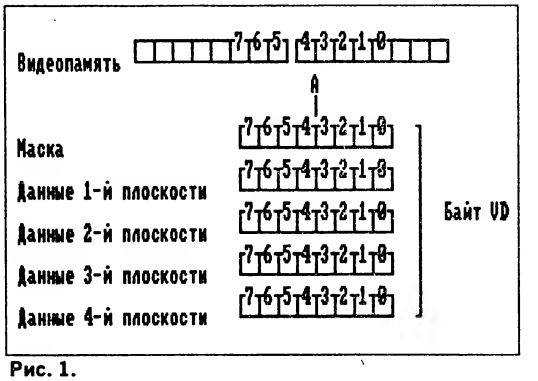


Рис. 1.

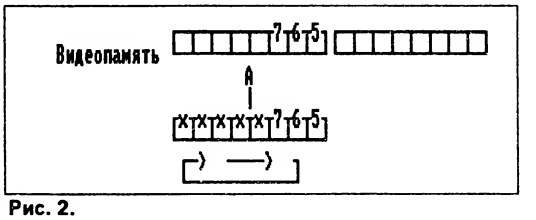


Рис. 2.

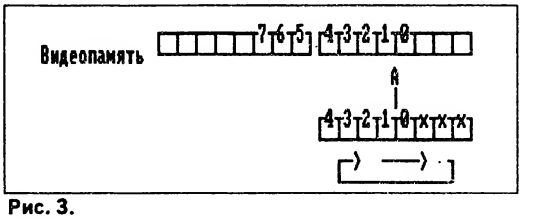


Рис. 3.

заполнены гораздо сильнее единиц, то будущее графическое изображение прекрасно видно на текстовом экране и для режима 640 на 350 пикселов сохраняется даже масштаб. Данный прием конечно не является заменой настоящего графического редактора, и в следующей статье будет рассказано о том, как получить из стандартного формата редактора данные, необходимые для работы описанной процедуры вывода. Причем, в программе EGASprite потребуется лишь заменить процедуру MakeSprite, а формат спрайта и собственно процедура вывода останутся неизменными. Прочитавший обе статьи станет обладателем полной технологической цепочки, соединяющей графический редактор с компьютерной игрой.

Листинг

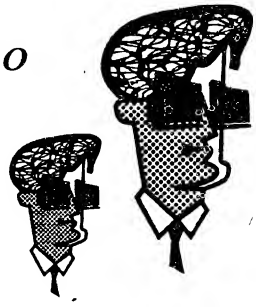
```
program EGASprite;
uses Graph, Crt;

const
  GD: Integer = 3;                    { Режим 640x350, 16 цветов}
  GM: Integer = 1;
  GraphScrSeg: Word = $A000;          { Сегментный адрес видеопамяти}
  ByteInString = 80;                  { Число байтов VD в строке экрана}
  SprLen = 3;                         { Длина спрайта}
  SprWid = 14;                        { Высота (ширина) спрайта}

type
  TByte = ^Byte;
  TBody = record                      { Структура для байта VD}
      Mask: Byte;                     { Маска для областей прозрачности}
      Plan: array [1..4] of Byte      { Данные для плоскостей}
  end;
  TSprite = record                    { Формат спрайта}
      Length: Byte;                   { Размер по X}
      Width: Byte;                    { Размер по Y}
      Body: array [1..SprLen*SprWid] of TBody    { Байты VD}
  end;

var
  Sprite: TSprite;
  X, Y, OldX, OldY, Vx, Vy: Integer;  { Координаты и скорости спрайта}
  Ch: Char;

procedure SetBit; assembler;          { Ставим маску битов, в AL - маска}
asm
  mov ah, al
  mov al, 8
  mov dx, 3CEh
  out dx, al
  mov al, ah
  inc dx
  out dx, al
end;

procedure SetWrMap; assembler;        { Ставим маску карт, в AL - маска}
asm
  mov ah, al
  mov al, 2
  mov dx, 3C4h
  out dx, al
  mov al, ah
  inc dx
  out dx, al
end;

procedure SetRot; assembler;          { Ставим вращение, в AL - вращение}
asm
  mov ah, al
  mov al, 3
  mov dx, 3CEh
  out dx, al
  mov al, ah
  inc dx
  out dx, al
end;
procedure Sync; assembler;            { Ждем обратного хода луча}
asm
  mov dx, 3DAh
@@Wait:
  in  al, dx
  test al, 08h
  jz @@Wait
end;
procedure SetLeftMask; assembler;     { Устанавливаем маску для левой}
asm                                   { части байта VD}
  lodsb                               { Получаем данные для маски}
  shr al, cl                          { В CL - сдвиг, если крутить вправо}
  call SetBit                         { Ставим битовую маску}
end;
procedure SetRightMask; assembler;    { Устанавливаем маску для правой}
asm                                   { части байта VD}
  lodsb                               { Получаем данные для маски}
  xchg ch, cl                         { В CH - сдвиг, если крутить влево}
  shl al, cl
  xchg ch, cl                         { Восстанавливаем CH и CL}
  call SetBit                         { Ставим битовую маску}
end;

procedure WriteWithMask; assembler;   { Пишем данные во все 4 плоскости}
var                                   { видеопамяти}
  PlnCnt: Byte;
asm
  mov al, byte ptr es:[bx+di]         { Заполняем регистр задвижки}
  mov PlnCnt, 00000001b               { Ставим счетчик плоскостей в 1}
@@NextPlan:                           { Цикл по 4 плоскостям}
  mov al, PlnCnt                      { Устанавливаем плоскость}
  call SetWrMap
  lodsb                               { Получаем данные для плоскости}
  mov byte ptr es:[bx+di], al         { Пишем данные в видеопамять}
  shl PlnCnt, 1                       { Следующая плоскость}
  cmp PlnCnt, 00010000b               { Проверяем конец цикла}
  jl @@NextPlan
end;

procedure PutSprite (X, Y: Integer; SprPointer: Pointer); assembler;
var
  SprLength, SprWidth,
  LengthCount, WidthCount: Byte;
asm
  push ds                             { Сохраняем DS}
  mov es, GraphScrSeg                 { В ES - сегмент экрана.}
  mov bx, Y                           { В BX - смещение, отслеживающее Y }
```

```
  mov ax, ByteInString
  mul bx
  mov hx, ax
  lds si, SprPointer              { DS:SI - указатель на данные спрайта}
  mov ax, X                       { Находим число целых байтов и сдвиг}
  mov cl, 8                       { внутри байта, отслеживающих X}
  div cl
  mov cl, ah
  xor ah, ah
  push ax                         { Сохраняем в стеке число целых байтов}
  mov ch, 8                       { В CH будет сдвиг, если крутить влево, }
  sub ch, cl                      { в CL - вправо}
  mov al, cl                      { Устанавливаем вращение}
  call SetRot
  lodsb                           { Считываем длину спрайта}
  mov  SprLength, al
  lodsb                           { Считываем высоту спрайта}
  mov SprWidth, al

  mov WidthCount, 0               { Цикл по Y}
  @@NextString:
  pop di                          { В DI - число целых байтов от начала}
  push di                         { строки (X)}
  mov LengthCount, 0              { Цикл по X}
    @@NxtByte:
    call SetLeftMask;             { Ставим маску для левой части байта VD}
    call WriteWithMask;           { Пишем данные с этой маской}
    sub si, 5                     { Возвращаемся на байт VD назад}
    inc di                        { Следующий байт на экране}
    call SetRightMask;            { Ставим маску для правой части байта VD}
  call WriteWithMask              { Пишем данные с этой маской}
  inc LengthCount                 { Следующий байт VD}
  mov al, LengthCount
  cmp al, SprLength               { Проверяем конец строки}
  jl @@NxtByte
  add  bx, ByteInString           { Следующая строка}
  inc WidthCount
  mov al, WidthCount
  cmp al, SprWidth                { Проверяем конец спрайта }
  jl @@NextString                 {                          }

  pop ax                          { Поправляем стек}
  mov al, 0                       { Убираем вращение}
  call SetRot
  mov al, 15                      { Убираем маску карт}
  call SetWrMap
  mov al, 255                     { Убираем маску битов}
  call SetBit
  pop ds                          { Возвращаем DS}
end;

procedure Man; assembler;         { Данные для маски битов  }
asm  { Массив данных 3x14}
  db 00000000b,01111110b,00000000b
  db 00000001b,10011001b,10000000b
  db 00000001b,11111111b,10000000b
  db 00000001b,10000001b,10000000b
  db 00000000b,11100111b,00000000b
  db 00000000b,00111100b,00000000b
  db 00000000b,11111111b,00000000b
  db 00000111b,11111111b,11100000b
  db 00011111b,11111111b,11111000b
  db 00111100b,11111111b,00111100b
  db 00000000b,11100111b,00000000b
  db 00000000b,11100111b,00000000b
  db 00000000b,11100111b,00000000b
end;

procedure MakeSprite;             { Заполняем данные для спрайта}
var
  i, j: Byte;
begin
  with Sprite do begin
      Length := SprLen;
      Width := SprWid;
      for i:= 1 to SprLen*SprWid do
        with Body[i] do begin
        { Обращаемся к процедуре, как к массиву данных      }
        {                                                   }
        {                                                   }
        Mask := TByte (LongInt (@Man) - 1 + i)^;
        Plan[1] := 0;
        for j:= 2 to 4 do Plan[j] := $FF;
      end;
  end;
end;

begin
  InitGraph (GD, GM, '');
  SetFillStyle (7, 1);
  Bar (0, 0, 639, 349);
  MakeSprite;
  X := 150; Y:= 275; Vx := 2; Vy := 1;
  repeat
      OldX := X; OldY := Y;
      X := X + Vx; Y := Y + Vy;
      if (X < 100) or (X > 540) then Vx := -Vx;
      if (Y < 250) or (Y > 335) then Vy := -Vy;
      Sync;
      Bar (OldX, OldY, OldX + SprLen*8, OldY + SprWid);
      PutSprite (X, Y, @Sprite);
  until KeyPressed;
  Ch := ReadKey;
  CloseGraph;
end.
```

Clipper или Paradox : попытка аргументированного выбора

Часть 2

С. Крапивенко



3. Представление данных

3.1. Поля таблиц

Представление полей таблиц в Paradox существенно удобнее и нагляднее, в свою очередь Clipper позволяет представлять данные более экономно (табл. 4 и 5).

Два типа данных имеют аналогичное в обеих системах представление: алфавитно-цифровые строки фиксированной длины от 1 до 255 символов (ALPHANUMERIC в Paradox и CHARACTER в Clipper) и даты (Date). В Clipper версии 5 длина строковых полей расширена до 32 Кбайтов.

Для представления чисел в Paradox существуют три типа данных: SHORT - двухбайтовые целые числа, NUMERIC - восьмибайтовые числа двойной точности, "$" - восьмибайтовые числа двойной точности для работы с денежными суммами (например 12,978,346.10).

Clipper поддерживает числовой тип данных с фиксированной точкой (NUMERIC). В определение типа входят два числа: первое определяет общее количество позиций, включая знак для отрицательных значений и десятичную точку, а второе - количество цифр мантиссы. Каждая позиция представляется одним байтом.

Таблица 4. Описание полей

Характеристика	Paradox	Clipper
Число символов имени поля	25	12
Разрешенные символы в именах полей	Любые символьные литеры	От 'A'до 'Z', подчеркивание '_'и дефис '-'
СИНТАКСИС ИМЕН ПОЛЕЙ: - поле текущей таблицы - текущее поле - поле таблицы - текущее поле таблицы - обращение к полю через значения переменных	Имя поля в квадратных скобках [] [<таблица> -> <поле>] [<таблица> ->] Только при позиционировании текущего поля	Как имя переменной Отсутствует <таблица> -> <поле> Отсутствует Допускается

Очевидно, что представление чисел в Clipper по сравнению с Paradox более экономичное (например : N 3.0 - целое число от -99 до 999, N 3.1 - числа от -.9 до 9.9 - занимают по 3 байта), но менее удобное, поскольку необходимо точно знать границы возможных значений и контролировать поля от переполнения или потери значащих цифр.

Кроме названных, в Clipper имеются дополнительно два типа полей - логическое поле и поле памяти. Логическое поле (LOGICAL) занимает 1 байт и

Таблица 5. Типы полей

Типы полей	Paradox	Clipper
Алфавитно-цифровые строки ограниченной длины	A<n> где <n> от 1 до 255	C<n> где <n> от 1 до 255 или до 32K (в. 5.0)
Дата	D	D
Логическое значение	Отсутствует	L
Короткое целое (2 байта)	S	N2.0
Число двойной точности	N	Отсутствует
Число для финансовых расчётов	$	Отсутствует
Число с фиксированной точкой	Отсутствует	N<n>.<m> где <n> — общее число позиций, <m> — для мантиссы
Поле памяти (до 64Кбайтов)	Отсутствует	M

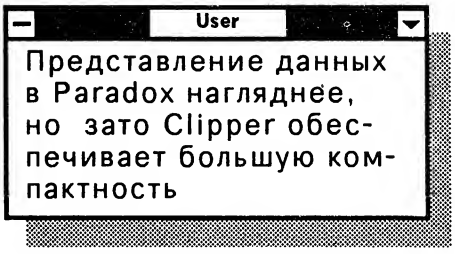
хранит одно из двух значений - T (истину) и F (ложь). Поле памяти (MEMO) используется для представления текстовых фрагментов значительной длины - до 64 Кбайтов. Значения этого типа хранятся в отдельном для каждого поля таблицы файле блоками по 512 байтов. Для работы с MEMO полями имеются специальные команды и функции.

Если отсутствие в Paradox логического типа можно компенсировать типом A1 (одним символом), то ограничение строковых полей 255 символами в ряде случаев вызывает большое неудобство. В свою очередь Paradox предоставляет русскоязычным разработчикам неоценимую услугу - русские буквы и прочие символы в именах полей, причем для Paradox эта возможность более значима, чем для Clipper ввиду жесткого интерфейса первого.

3.2. Переменные

Обе системы поддерживают скалярные переменные и одномерные массивы перечисленных типов данных. Тип переменной определяется типом значения выражения в операторе присваивания и может быть переопределен многократно. Массивы могут включать элементы различных типов, а размер массива может переопределяться. Существуют глобальные и локальные переменные. Первые доступны во всех подпрограммах, последние - только в текущих.

В Paradox локальные переменные обозначаются словом PRIVATE, для объявления глобальных переменных особых действий не предусмотрено. В закрываемых (CLOSED) процедурах внешние переменные доступны или через строку параметров, или с помощью специального объявления (раздел USEVARS). К сожалению, в Paradox массивы - только глобальные. Как следствие массив не может быть передан в процедуру в качестве параметра. Для обхода этой трудности можно предложить следующий прием : в качестве параметра передается



текстовая строка с именем массива, а в теле процедуры каждое обращение к элементу массива осуществляется с помощью оператора EXECUTE, который выполняет (интерпретирует) текстовую строку.

В свою очередь Paradox имеет дополнительные возможности использования массивов при работе с таблицами: оператор CopyToArray <имя_массива> формирует массив и сохраняет в нем значения полей текущей записи в порядке их следования, оператор CopyFromArray <имя_массива> переписывает значения элементов массива в поля текущей записи, а дополнительный элемент массива содержит информацию об именах и типах полей, позволяющую обращаться к элементам массива как к именам полей.

Clipper поддерживает переменные и глобальные (Public) и локальные (Private) массивы, в версию 5.0 добавлены Local и Static объекты. Local переменные, как и Private, скрыты внутри процедур, но представляют собой адреса памяти, что несколько экономит память за счет сокращения таблицы символов, но не позволяет использовать их в макроподстановках. Static переменные не теряют своих значений при выходе из подпрограмм. Как показали эксперименты, Private переменные также сохраняют свои значения, хотя в документации об этом не сообщается, а для "очистки" необходимо употреблять команду RELEASE с параметром ALL.

В Clipper 5.0 разрешено построение многомерных массивов путем объявления элемента массива как массива и так далее, что позволяет строить "деревья" причудливой формы.

3.3. Идентификаторы как параметры команд

При написании команд с параметрами в ряде случаев (и в рассматриваемых СУБД, и в прочих) требуется помнить, какие параметры воспринимаются в качестве константы, а какие - в качестве выражения (частный случай выражения - идентификатор переменной).

В Paradox, как правило, параметры команд задаются выражениями, т.е. просто отличить (и использовать) идентификаторы, текстовые и числовые константы, а также выражения. Есть все же несколько исключений. При работе с массивами и процедурами (определение, употребление и уничтожение массивов, запись процедуры в библиотеку или чтение ее из библиотеки и т.п.) требуются идентификаторы. Например, в команде записи процедуры в библиотеку (WRITELIB <имя библиотеки> <имя процедуры>) в качестве имени библиотеки должно быть выражение, а в качестве второго параметра - идентификатор. Или обращение к полю открытой таблицы, не являющейся текущей ([<имя таблицы> - <имя поля>]) в качестве имен таблицы и поля также предполагает идентификаторы. Обойти подобные ограничения можно с помощью команды EXECUTE с одним параметром – текстовой строкой, представляющей собой ЛЮБОЙ правильный оператор для его выполнения (интерпретации).

В Clipper параметры ряда команд должны задаваться идентификаторами либо макроподстановками переменных. Макроподстановки позволяют вместо имен или констант употреблять переменные, содержащие значения соответствующих контексту типов. Записывается макроподстановка в виде знака "&" (амперсэнд) слева от идентификатора без промежуточных пробелов. Возможны конкатенации макроподстановок, например: переменная VAR1 содержит значение "MY", а переменная VAR2 - "_TABLE"; макроподстановка &VAR1&VAR2 даст строку MY_TABLE. Сфера действия макроподстановок все же значительно уже оператора EXECUTE в Paradox.

4. Техника работы с таблицами

4.1. Работа со структурой таблицы

4.1.1. Создание и корректировка структуры таблицы

В Paradox структура таблицы может быть создана на основе структуры имеющейся таблицы, как результат импорта таблицы из другой СУБД, и путем заполнения таблицы структуры, состоящей из двух полей: поля имен и поля типов полей.

Изменение структуры таблицы осуществляется путем корректировки таблицы структуры. И ввод, и корректировка структуры выполняются как интерактивно, так и из программы. В последнем случае (и это типично для Paradox) в программе используются команды - эквиваленты клавиш передвижения (HOME, END, LEFT и т.д.), стирания (BACKSPACE, CTRLBACKSPACE), а для добавления к содержимому поля чего-либо употребляется оператор TYPEIN (выражение) или просто "строка_ввода". В приведенном ниже фрагменте программы:

```
{Create} {my_table} "First field"
Right "a5" Do_It!
```

создается таблица с именем "my_table", одним полем "First field" типа "алфавитно-цифровая строка" с числом символов не более пяти, неиндексированная. Команда создания таблицы CREATE имеет также и "неинтерактивный" вариант:

```
Create "My_table" "Первое поле" :
"a5", "Второе поле" : "n";
```

В первом варианте Paradox проверяет, есть ли таблица с таким же именем, и если да, выдает сообщение и меню из двух пунктов - "Cancel" и "Replace". В программе можно проверить существование на диске файла с этим именем или добавить проверку обработки в месте возникновения предупреждения, т.е. после указания имени таблицы. Если функция MenuChoice () возвратит значение "Cancel" - значит имя таблицы дублированно.

Чтобы при создании новой таблицы взять за основу структуру другой таблицы, после команды CREATE поставьте слово LIKE.

Для корректировки структуры таблицы в Paradox имеется только "интерактивный" вариант: {Modify} {Restructure} <имя_таблицы> и далее команды корректировки как для обычной таблицы. При завершении корректировки по клавише подтверждения (DO_IT!) значения полей преобразуются в соответствии с новыми типами данных, насколько это возможно. Записи, при преобразовании которых какие-либо значения полей "потерялись", копируются в таблицу с именем "PROBLEMS", структура которой совпадает с исходной. При этом "PROBLEMS" будет открыта автоматически.

В Clipper структура таблицы может быть создана интерактивно утилитой DBU, а в программе - на основе структуры другой таблицы с возможностью отбора части полей (COPY STRUCTURE TO<имя новой таблицы> FIELDS список полей) или вводом новой структуры. Для ввода новой структуры необходимо выполнить три шага: создать пустую таблицу структуры (CREATE <имя таблицы структуры>) из четырех полей - FIELD_NAME, FIELD_TYPE, FIELD_LEN и FIELD_DEC, затем открыть ее в свободной области, заполнить значениями и закрыть, а в завершение - создать новую таблицу (CREATE FROM <имя таблицы структуры>). Все имена новых таблиц должны быть уникальными, иначе возникнет ошибка.

Для корректировки структуры таблицы в программе необходимо записать структуру в таблицу (COPY TO <имя таблицы структуры> EXTENDED), открыть ее, откорректировать, закрыть, удалить или переименовать старую таблицу и затем создать новую структуру (CREATE FROM ...). При этом все заботы по восстановлению значений из старой таблицы возлагаются на программиста.

4.1.2. Анализ структуры

В Paradox для получения структуры таблицы в программе надо ввести:

```
{Tools} {Info} {Structure} select (name)
```

Будет создана и открыта таблица структуры с именем STRUCT, состоящая из двух текстовых полей - "Field Name" и "Field Type".

Примечание. В Paradox для указания объекта (аргумента какой-либо команды из программы) существуют несколько вариантов. Если объект - константа, заключите константу в фигурные скобки ({my_table} - см. пример из пункта 4.1.1) или в апострофы - всю константу или только первую букву (для выбора пункта меню). Если значение задается переменной или выражением, нужен оператор SELECT (выражение) или две команды: TYPEIN (выражение) ENTER. В интерактивном режиме при выборе альтернатив меню достаточно нажимать первую букву требуемого пункта, а для указания конкретного значения (скажем, имени результирующей таблицы) ввести это значение (если объект уже существует) либо нажать клавишу ENTER для появления всего списка объектов (таблиц, форм и т.д. - в зависимости от контекста), затем ограничить список начальной буквой, и если таких элементов несколько, то уже из них выбрать клавишами перемещения курсора требуемый объект и нажать ENTER.

В Clipper существует функция AFIELDS (), возвращающая количество полей текущей таблицы и заполняющая несколько массивов атрибутами полей: именами полей, типами, размерами, длинами дробной части полей числового типа. Можно задействовать все четыре массива или некоторые из них; для пропуска элементов списка ставьте переменные с пустыми значениями. Перед тем как заполнить массивы, необходимо объявить их размер (не меньший количества полей таблицы).

Например, чтобы получить имена и длины полей таблицы в текущей области, используйте следующий фрагмент:

```
DECLARE NAMES [FCOUNT ()], LENGHTS [FCOUNT ()]
DUMMY = ""
AFIELDS (NAMES, DUMMY, LENGHTS)
```

Заметим, что в отличие от Paradox, в Clipper доступ к структуре возможен только для открытой таблицы в текущей области.

4.2. Импорт и экспорт

Paradox поддерживает импорт и экспорт данных наиболее распространенных форматов. Для того чтобы преобразовать, к примеру, таблицу в формате СУБД dBase (версий II, III, III Plus, IV) в таблицу Paradox, последовательность выбора пунктов меню такова:

```
{Tools} {ExportImport} {Export} {Dbase}
select (dBaseTable) select (ParadoxTable)
```

Полный состав меню экспорта/импорта Paradox следующий:

```
"Quattro/PRO", "1-2-3", "Symphony", "Dbase", "Pfs",
"Reflex", "Visicalc", "Ascii".
```

При импорте из традиционного Ascii - формата текстовых файлов возможны три варианта: "Delimited", "AppendDelimited", "Text". Первые два варианта предполагают отделение значе-

ний полей в текстовом файле запятыми, а для текстовых полей - еще и апострофами. Первый вариант создает новую таблицу с предопределенными именами полей; второй - использует уже имеющуюся структуру таблицы и не портит ее содержимого. Последний вариант (Text) создает таблицу из одного текстового поля.

При экспорте данных из таблицы Paradox в формат Ascii файлов предлагаются варианты "Delimited" и "Text".

Clipper предоставляет небогатый выбор - только Ascii файлы, но двух видов и с некоторыми вариациями.

Полный формат экспорта:

```
COPY TO <имя файла>
[область действия] [FIELDS <список полей>] [условие выбора]
{ [SDF] | DELIMITED [WITH
{BLANK | символ-ограничитель } ] }
```

где имя файла - результат, область действия - все (ALL - по умолчанию) или несколько записей (REST - до конца таблицы, NEXT <n> - следующие <n> записей), список полей - интересующие поля, условие выбора : FOR-условие отфильтровывает записи из всех доступных, а ложность WHILE-условия определяет останов; SDF - формат данных, в котором результирующая строка совпадает с содержимым записи базы данных плюс по одному пробелу - разделителю полей, DELIMITED - формат данных, в котором текстовые поля заключаются в апострофы, все поля отделяются символом - ограничителем, BLANK - означает, что в качестве разделителя употребляется пробел, символ-ограничитель определяет символ для разделения значений полей, если он не совпадает с символом <,> (запятая) (по умолчанию). В прямоугольные скобки заключены необязательные параметры, в фигурные - варианты, разделителем которых служит символ "|".

Команда импорта - APPEND FROM имеет все те же параметры, что и команда COPY TO.

4.2.1. Вывод данных в текстовый файл

Кроме экспорта содержимого таблиц в формат Ascii обе системы поддерживают вывод информации непосредственно в текстовый файл.

В Paradox - два варианта. Первый заключается в определении вывода для формы отчета как одной из трех альтернатив (экран, принтер, файл). Второй - по оператору PRINT FILE. Формат этой команды:

```
PRINT FILE <имя файла><список выражений>
```

Список выражений разделяется запятыми; запись заканчивается Ascii - символом LineFeed (десятичный код 10). Каждый оператор вывода добавляет данные к содержимому файла. Файл открывается при первом обращении по команде вывода и держится открытым насколько это возможно, затем автоматически закрывается.

В Clipper вывод в текстовый файл осуществляется посредством дублирования вывода информации кроме экрана в файл по команде SET ALTERNATE. Файл вывода открывается командой SET ALTERNATE TO <имя файла>; закрывается - той же командой без указания имени файла или CLOSE ALTERNATE. Начало вывода информации происходит после команды SET ALTERNATE ON, окончание - SET ALTERNATE OFF. Дублирование вывода информации на экран можно подавить командой SET CONSOLE <логическое выражение или ON/OFF>. Команды дисплейного вывода

```
@<строка>,<столбец>SAY<выражение>
```

не влияют на альтернативный вывод.

4.3. Сортировка

Для выполнения сортировки записей в Paradox предназначена команда SORT. Интерактивный вариант этой команды

```
{Modify} {Sort} select (имя таблицы)
```

(в отличие от многих других команд) не позволяет моделировать нажатие клавиш вплоть до того момента, как пользователь нажмет клавишу завершения ввода параметров сортировки. Поэтому остановимся на "пакетном" варианте:

```
SORT таблица-источник ON <список имен полей>
[TO таблица-результат],
```

где список имен полей - это имена полей в апострофах, разделенных запятыми, определяющих порядок сортировки. Если требуется сортировка по убыванию значения, сразу за именем соответствующего поля поставьте символ D (Descending). Таблица-результат указывается, если результат сортировки - не исходная таблица. Этот вариант обязателен для индексированных таблиц, поскольку для них порядок следования записей соответствует индексному ключу (см. п.4.4.).

Формат команды сортировки в Clipper:

```
SORT [область] ON <список полей>
TO <таблица-результат> [условие выбора]
```

где область и условие выбора - см. команду COPY TO (п. 4.2.), список имен полей - это имена полей (заданные константами или макроподстановками), разделенные запятыми, определяют порядок сортировки. Если требуется сортировка по убыванию значения, сразу за именем соответствующего поля ставится параметр /D, по возрастанию - /A, и с игнорированием различий между строчными и прописными буквами - /C (последний можно использовать вместе с одним из /D или /A). Таблица - результат не может совпадать с какой-либо из существующих таблиц.

4.4. Индексирование

Как известно, индексирование в базах данных применяется в следующих целях: быстрый поиск (не более одного - двух обращений к диску), изменение порядка следования записей без пересортировки путем указания нового индексного выражения, а также установления связей между таблицами (Relation) для автоматического передвижения указателей текущих записей в зависимых таблицах при перемещении указателя текущей записи в главной таблице.

И в Paradox, и в Clipper имеются средства для поддержки индексов, но характер действий по использованию индексов отличается значительно.

Индексное выражение в Paradox состоит из последовательности значений первых полей таблицы и задается при создании или изменении структуры таблицы путем добавления к типу данных знака "*" (звездочка). Имя индексного файла, как и всех файлов семейства, совпадает с именем таблицы, расширение - ".PX", все семейство открывается автоматически.

Поиск записей в индексированных таблицах ничем не отличается от поиска в неиндексированных. Для поиска по одному полю необходимо

переместиться в интересующее при поиске поле (MOVETO FIELD имя_поля) и выполнить команду поиска (значение LOCATE либо значение LOCATE NEXT для поиска соответственно от начала таблицы или с текущей записи). При поиске по первым полям формат команды следующий :

```
LOCATE [NEXT] значение_1,
значение_2, ... значение_N.
```

Системная переменная RetVal возвращает результат операции (True/False).

Порядок записей в индексированных таблицах в Paradox определяется значением индексного ключа и не может быть изменен без изменения индексного выражения.

Связи между таблицами устанавливаются через ввод примеров (EXAMPLE) - совпадающих значений полей - в команде запроса (Ask/Query) и при формировании отчета (Report), а также через многотабличные экранные формы. В последнем случае любая из четырех связей - "один к одному", "один ко многим", "многие к одному", "многие ко многим" организуется в соответствии с составом индексных полей в главной и зависимой таблицах и составом индексных полей в связующих формах. С помощью генератора экранных форм необходимо построить форму, содержащую одно или несколько полей зависимой таблицы, и включить эту форму в форму главной таблицы. Связующая форма активизируется командой PickForm "F" ("стандартная" форма) или PickForm "Fномер_формы", где номер_формы - число от 1 до 14.

Все вышесказанное об индексах Paradox относилось к главному индексу. Paradox также поддерживает вторичные индексы (Secondary Indexes), основное назначение которых - оптимизация запросов ("Ask"). При выполнении запроса система при необходимости автоматически формирует вторичные индексы или использует имеющиеся, сохраненные ранее командой оптимизации

```
MENU {Tools} {QuerySpeed}
```

либо командой индексирования:

```
INDEX имя_таблицы ON имя_поля.
```

При этом для каждого поля создаются по два файла с расширениями "Xnn" и "Ynn", где nn - номера индексов, начиная с "01".

Clipper для каждой таблицы поддерживает неограниченное число индексных файлов (до 255). Индексное выражение может быть любого допустимого типа данных (кроме MEMO) и обычно включает значения полей таблицы или результат выполнения встроенных функций Clipper с полями таблицы в качестве параметров. Размер индексного выражения - не более 255 символов. Индекс может быть создан по мере надобности командой INDEX ON индексное_выражение TO индексный_файл. Имя файла - любое, расширение - NTX. Одновременно могут быть открыты все созданные для таблицы индексные файлы (это необходимо для обновления индексов по мере изменения данных в таблице).

В конкретный момент один из индексов - текущий, он определяет порядок следования записей и формулу индекса для быстрого поиска или связи с главной таблицей. После открытия индексов первый в списке индексный файл становится текущим. Выбор текущего индекса осуществляется командой SET ORDER TO номер_файла_в_списке. Для оперативного управления текущим индексом существуют функции IndexKey (номер_индекса) и IndexOrder (значение_индексного_выражения), возвращающие соответственно значение индексного выражения для текущего или указанного порядковым номером файла и порядковый номер файла для текущего индекса или указанного индексного выражения.

Быстрый поиск по индексному ключу осуществляется командой SEEK значение; результат операции - логическое значение функции FOUND (). Возможен поиск по точному совпадению с указанным значением или по частичному - в соответствии с установкой режима командой SET EXACT ON или SET EXACT OFF. Кроме того, при неудачном поиске возможно позиционирование либо на записи с индексным ключом, большем чем указанный, либо - за последней записью - в соответствии с установкой SET SOFTSEEK ON / OFF.

Связь между таблицами устанавливается одним из трех способов - по значению индексного выражения, по номеру записи и по числовому выражению для вычисления номера записи. В последних двух вариантах зависимая таблица не должна иметь открытых индексов. Чаще используется первый вариант :

```
SET RELATION TO индексное_выражение
INTO зависимая_таблица.
```

Важно отметить, что в Clipper можно установить связь не только главной таблицы с несколькими зависимыми, но и зависимые с зависимыми от них, т.е. "связь по цепочке".

(Продолжение в следующем номере)

Монитор – 1993

Подписной купон

МОНИТОР

Адрес _____

_____

Ф.И.О. или название _____

организации _____

Стоимость подписки на 1 год (10 номеров) – 600 рублей. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г.Москве, МФО 201791.

Вышлите подписной купон и квитанцию об оплате по адресу: 117419, Москва, а/я 765.

Что такое Clarion

Л. Ливеровский

В последнее время в нашей стране возрос интерес к инструментальному интегрированному пакету Clarion. В среде программистов бытуют диаметрально противоположные мнения о возможностях этого программного средства, начиная от мнения, что Clarion чуть ли не вообще избавляет программиста от необходимости кодирования своего приложения, и заканчивая суждением, что Clarion не стоит никакого внимания программиста-прикладника.

Так что же такое Clarion ?

Clarion - это интегрированная среда (система) программирования (ИСП), предназначенная для проведения полного цикла разработки коммерческого заказного программного обеспечения (ПО) в области информационно-поисковых систем (ИПС) и "деловых" приложений на ПЭВМ типа IBM, работающих под управлением ОС MS-DOS.

Она ориентирована на широкий круг пользователей от профессионалов- разработчиков прикладных систем до тех, для кого программирование является лишь дополнительным инструментом, используемым в работе.

Система Clarion была разработана фирмой Clarion Software Corporation в 1988 году. В 1992 году должна выйти версия 3.0 продукта. Данная статья базируется на версии 2.1.

Основными целями, преследовавшимися разработчиками среды Clarion, по мнению автора статьи были:

- значительное сокращение времени и трудоемкости создания заказного ПО;
- обеспечение возможности участия в разработке заказного ПО самого заказчика, как правило не являющегося специалистом в программировании;
- освобождение разработчика от рутинной работы по разработке экранов и отчетов и представление ему возможности сконцентрироваться на проблемах прикладной области;
- обеспечение автоматизации ведения проекта (Clarion имеет больше возможностей, чем традиционная утилита ведения проекта MAKE);
- предоставление разработчику механизма структурного проектирования программы методом "сверху - вниз".

Все эти цели в достаточной мере были достигнуты благодаря удачному сочетанию набора подсистем, входящих в СП Clarion, и выполняемых ими функций.

Основу системы составляют язык программирования универсального типа Clarion с некоторыми интересными особенностями и файловая система, поддерживаемая средствами языка.

К этим особенностям языка следует отнести, во-первых, описание плана программы (файлов с исходными текстами, процедур, функций и оверлеев), задаваемое специальным декларативным оператором MAP, помещаемым в головную программу. Это позволяет компилятору транслировать только те модули, в которые вносились изменения, и программисту обойтись без специальной утилиты ведения проекта MAKE, имеющейся в других системах программирования.

Во-вторых, в языке имеются специальные структуры данных SCREEN и REPORT, описывающие формат экрана и поля ввода/отображения и формат отчета для печати.

В-третьих, в языке имеется возможность с помощью специального типа данных TABLE работать с памятью так же, как с файлом.

Файловая система позволяет работать с двумя видами файлов:

- с файлами ДОС!
- с файлами Clarion. Файлы Clarion - это наборы записей фиксированной длины. Доступ к записям может осуществляться последовательно по номеру записи и по ключам.

Систему управления файлами Clarion нельзя назвать реляционной СУБД, т.к. она не поддерживает операции реляционной алгебры: объединение, проекцию, селекцию, разность и декартово произведение таблиц или, говоря математическим языком, множеств кортежей. Однако эти операции могут быть достаточно просто запрограммированы на языке Clarion.

Файловая система Clarion поддерживает защиту данных от несанкционированного использования. Файлы могут иметь специальные атрибуты "владелец" и "пароль", разрешающие доступ к данным только после указания правильного пароля и владельца. Сами данные при этом хранятся в файлах в зашифрованном виде. В системе имеются средства поддержки транзакций.

Большим достоинством для отечественного пользователя этой системы является то, что с символами кириллицы Clarion работает без всякой "русификации" - то есть везде, где используются символьные строки, они могут содержать любые символы кириллицы.

Функции перевода символов из одного регистра в другой естественно работают с кириллицей неправильно. Однако они без особого труда могут быть переписаны.

Более того много сообщений и некоторые настройки исполняющей системы Clarion хранятся в специальном файле OPTIONS.CTL, и имеется специальная программа OPTIONS, с помощью которой можно поменять тексты сообщений, сокращения названий месяцев, порядок следования символов в кодовой таблице (для сортировок) и таблицу настройки функций перевода знаков алфавита из одного регистра в другой. К сожалению, размер этой таблицы не позволяет включить в нее все символы кириллицы (ее размера хватает только до "Ч").

Вся система использует только символьный режим видеоадаптеров и не конфликтует ни с одним известным автору драйвером-"русификатором" ПЭВМ.

В состав системы входят следующие основные компоненты, объединенные общей оболочкой:

- двухоконный синтаксически ориентированный редактор с встроенной подсистемой поддержки разработки диалога (Editor). Например, для создания описания макета экрана (оператор SCREEN) по нажатию определённой клавиши редактор переходит в режим "визуального" проектирования экрана. Пользователю остается нарисовать необходимые рамки, написать в нужных местах текст, определить необходимые поля: отображения, ввода, меню. После этого описание экрана встраивается непосредственно в текст программы;
- генератор отчетов (Reporter);
- построитель подсказок (помощи) (Helper). Помощь строится по принципам гипертекста;
- компилятор (Compiler) компилирует исходные модули в некий промежуточный код. Откомпилированные модули имеют расширение .PRO;
- интерпретатор с отладчиком (Processor). Интерпре-

тируются модули в промежуточном коде;

- транслятор (Translator) транслирует модули из промежуточного кода (.PRO) в стандартные объектные модули MS-DOS (.OBJ), предназначенные для последующей компоновки одним из компоновщиков (Link, PLINK86, RTLink) и создания стандартной выполняемой программы MS-DOS (.EXE);
- импорт-экспорт данных из файлов (Converter). Поддерживаются форматы файлов dBASE, DIF, BASIC;
- работа с файлами Clarion (редактирование, добавление, удаление) (Scanner);
- сортировка файлов Clarion (Sorter);
- создание/изменение структуры файлов Clarion (Filer).

Все эти программы также могут выполняться в пакетном режиме из командной строки ДОС.

В состав пакета входит и обучающая программа.

Особое место в ИСП Clarion по праву занимает генератор приложений или проектировщик (Designer).

С одной стороны, он позволяет создавать простые приложения с самого начала (проектирования баз данных) до самого конца (получения готового приложения в виде выполняемой программы). При этом знание языка программирования является необязательным.

С другой стороны, специалисту, разрабатывающему более сложные программы, дизайнер поможет быстро создать прототип всей системы, спроектировать структуру программы, экраны и отчеты, отследить целостность проекта, и тем самым позволит большую долю времени уделить собственно содержательной части прикладной задачи.

Из перечисленных компонент интегрированной системы программирования наибольший интерес представляют язык программирования и генератор приложений или проектировщик. Рассмотрим их подробнее.

Язык Clarion
Типы данных и переменные

Основные скалярные (простые) типы данных:

BYTE - 1-байтовое беззнаковое целое;
SHORT - 2-байтовое целое со знаком;
LONG - 4-байтовое целое со знаком;
REAL - 8-байтовое число с плавающей точкой;
DECIMAL - упакованное десятичное со знаком и дробной частью (длина 8 байтов или 15 десятичных цифр);
STRING - строка длиной до 255 байтов.

В языке не существует типов данных "логический" и "дата". Тип данных "логический" моделируется с помощью целого (0 - ложь, не 0 - истина). Даты хранятся в виде длинных целых. В языке предусмотрены специальные средства для работы с длинными целыми, как с датами.

Основные агрегатированные (сложные) типы данных

Массив - объединение однотипных элементов.

Размерность массивов не может превышать 4. Общий размер массива не может превосходить 65520 байтов.

Например:

```
MATRIX SHORT,DIM(4,8)
```

Структура - объединение разнотипных элементов

Например:

```
PERSON GROUP        !ЧЕЛОВЕК
NAME   STRING(32)   !Ф.И.О.
AGE    DECIMAL(3,0,0)!Возраст.
SEX    STRING(3)    !Пол 'муж,'
                    !'жен'.
                    !Точка.
                    !Является элементом
                    !синтаксиса. Может быть
                    !заменена на ключевое
                    !слово END
```

"!" - признак комментария.

Структуры могут быть вложенными. Могут существовать массивы структур.

Ссылка на элемент структуры осуществляется непосредственно по имени этого элемента. В связи с этим все имена переменных, массивов, структур, элементов структур и т.д. в программе, процедуре или функции должны быть уникальными, что не очень удобно. Для того чтобы избежать этого неудобства, к структуре, файлу, записям в файлах, экранам, отчетам может быть добавлен префикс с помощью специального атрибута объявления PRE(<префикс>), и далее можно ссылаться на элементы этих структур с его помощью:<префикс>:<имя>.

Как видно, в языке Clarion отсутствуют такие типы данных, как указатели или ссылки.

Память под переменные выделяется на этапе трансляции, и размер данных не может изменяться в процессе выполнения программы. В языке нет механизмов динамического выделения памяти под произвольные объекты.

Константы операции и выражения

В языке существуют константы следующих видов:

двоичные (00101111b),восьмеричные (57o), шестнадцатиричные (5Fh). десятичные (47), действительные (4.7e+0), строковые ('ABCDE').

Набор операций в языке Clarion стандартен. Существуют арифметические операции, строковые операции, операции сравнения, логические операции.

Выражения в языке строятся по обычным правилам с использованием стандартного для всех языков старшинства операций. В выражениях могут использоваться вызовы функций в любом месте, где допускается использование переменной. При вычислении выражений производится автоматическое преобразование типов.

Основные операторы языка

Операторы бывают простые (оператор присваивания) или структурные (условный оператор).

Операторы могут иметь метки. Метка должна начинаться в первой позиции. Все операторы начинаются не раньше второй позиции.

Операторы, управляющие выполнением программы

В этой группе операторов присутствуют все обычные операторы, характерные для универсальных языков программирования: оператор перехода GOTO, операторы циклов с проверкой условия в начале цикла (LOOP WHILE), с проверкой условия в конце цикла (LOOP UNTIL), оператор цикла со счетчиком шагов (LOOP TIMES), условный оператор (IF-THEN-ELSE), операторы выбора из нескольких возможностей (CASE и EXECUTE - аналог вычисляемого GOTO в ФОРТРАНе или оператора ON .. . GOSUB в БЕЙСИКе.)

Подпрограммы и функции

В языке Clarion существуют несколько типов подпрограмм: локальная процедура (local routine) вызывается оператором DO; процедура вызывается оператором вызова процедуры; функция вызывается в любом месте, где допустимо использование переменной.

Процедуры и функции, написанные пользователем или присутствующие в модулях расширения языка (LEM - Language Extension Modules), требуют специального описания в головной программе оператора MAP.

Процедуры и функции, встроенные в язык, не требуют специального описания в головной программе. Язык содержит богатый набор встроенных функций (более 100).

Общая структура программы

Программа на языке Clarion обязательно содержит главную программу PROGRAM. В ней в операторе MAP описывается вся структура программы и содержатся все глобальные данные. Ссылки на глобальные данные разрешаются на этапе компиляции. Поэтому изменения в главной программе требуют перекомпиляции всех других модулей.

Во всех исходных файлах программы, кроме файла с головной программой, в операторе MEMBER необходимо указать имя файла, содержащего головную программу.

В связи с тем, что в операторе MEMBER необходимо явно указать главную программу (а это необходимо, т.к . настройка адресов и разрешение внешних ссылок происходят на этапе компиляции), на языке Clarion невозможно создать библиотеку подпрограмм универсального назначения. Это несомненно является одной из "находок" проектировщиков языка. Более того к программе на языке Clarion невозможно напрямую подстыковать объектные модули, созданные на других языках программирования.

Невозможность создания библиотек подпрограмм на языке Clarion и подстыковки объектных модулей компенсируется наличием возможности присоединения к программе специальных двоичных файлов, называемых LEM, содержащих необхо-

димые подпрограммы, составленные на других языках (Си, Ассемблере и т.д.).

На этой основе создаются файлы LEM, содержащие различные наборы полезных функций и процедур. Для примера укажем на существование LEM для работы с файлами DBF, с файловой системой BTrieve фирмы Novell и для обслуживания последовательного интерфейса. Однако файл расширения языка не может быть больше 64K (так же, как и .COM файл ДОС).

Файлы Clarion

Описывается файл структурным оператором FILE.

Файл Clarion состоит из записей. Структура записи задается с помощью оператора RECORD, помещаемого внутри оператора FILE.

Типы полей могут быть такими же, как и типы переменных (включая массивы и группы). Количество полей в записи не ограничено. Максимальный размер записи 65520 байтов. Максимальное количество записей в файле в документации не указано. Однако операции доступа к записям по их номерам используют переменные типа LONG для указания номера записи, что накладывает ограничение в 2147483647 записей в файле, к которому осуществляется прямой доступ.

Записи считываются из файла в структуру RECORD командами NEXT (следующая), PREVIOS (предыдущая) и GET (получить). Записи записываются в файл командами PUT (записать), ADD и APPEND (добавить) и DELETE (удалить).

Файл Clarion может иметь ключи и индексы.

Описание ключей и индексов задается с помощью операторов KEY и INDEX, помещаемых внутри оператора FILE и перед оператором RECORD.

Записи могут иметь как простой ключ или индекс, состоящий из одного поля, так и сложный ключ, состоящий из нескольких полей. Все ключевые файлы обновляются синхронно с обновлением файла. Индексные файлы перед использованием необходимо обновить (если после последнего обновления файл данных менялся) командой BUILD.

В отличие от индексов dBASE, Cliper, FOXBase в индексах Clarion не может быть выражений, а может быть только список полей в порядке значимости для упорядочивания.

Максимальный размер ключа 245 байтов.

Файл Clarion фактически состоит из нескольких файлов MS-DOS: файла с данными (расширение .DAT), по одному файлу на каждый ключ и индекс (расширения .Kxx, .Ixx, где xx принимает значения от 00 до ZZ) и возможно файл с MEMO полями (расширение .MEM).

Из-за того, что структура файла и всех его ключей и индексов должна явно задаваться в теле программы, при внесении изменений в структуру файла необходимо изменить и перетранслировать все программы, использующие этот файл. Поэтому в языке Clarion трудно написать программу, обрабатывающую файлы произвольной структуры.

Одним из сильных средств языка Clarion по работе с файлами является система поддержки транзакций. Под транзакцией понимается логически неделимая операция, которая требует обновления нескольких записей в нескольких файлах. Для каждого файла Clarion, участвующего в транзакции, заводится так называемый "файл отката" (pre-Image log file), содержащий копии записей основного файла до их изменения.

Оператор LOGOUT (<файл>) инициирует регистрацию для транзакций. Параметр <файл> - это есть спецификация файла регистраций, который содержит спецификации всех файлов Clarion, которые были изменены с помощью операторов ADD, APPEND, DELETE, PUT с тех пор, как была начата регистрация. Когда выполняется оператор LOGOUT, создается или опустошается (если уже существовал) файл регистрации транзакций. Затем в течение регистрации для каждого изменяемого файла Clarion в файл регистрации записывается спецификация "файла отката", и копия каждой записи до ее изменения пишется в соответствующий файл отката.

Оператор COMMIT заканчивает текущую транзакцию. Оператор ROLLBACK восстанавливает состояние файлов, существовавшее до начала транзакции.

В языке имеются средства для работы с файлами в сетевой среде, такие как монопольная и разделяемая работа с файлом, блокировка/деблокировка записей и файлов.

Также имеются средства управления буферизацией и кешированием файлов.

Экраны

Экран - это особая структура в языке Clarion, описывающая макет экрана. Она, с одной стороны, является образом изображения, которое мы видим на экране видеомонитора, или копией содержимого видеопамяти видеоконтроллера, и содержит операторы, описывающие все символы, отображаемые на дисплее, и их видеоатрибуты.

С другой стороны, она содержит описания всех полей ввода. Существуют пять типов полей ввода:

- текстовое поле;
- числовое поле;
- пункт меню;
- пауза;
- поле указатель;

Экран описывается специальным структурным оператором SCREEN.

Оператор SCREEN содержит операторы, которые описывают символы на экране дисплея (STRING), видеоатрибуты (PAINT) и входные поля (ENTRY, TEXT, MENU, POINT И PAUSE).

Структура SCREEN открывается и становится активной при помощи оператора OPEN, а закрывается оператором CLOSE. Одновременно активной может быть только одна структура SCREEN.

SCREEN без атрибута WINDOW (описанного ниже) называется "базовым экраном". Когда базовый экран открывается, все другие открытые экраны закрываются.

Размер окна, определяемого атрибутом WINDOW(<рядов>,<колонок>), задается константными выражениями. Поэтому невозможно вычислять размеры окна в процессе выполнения программы, и затем активизировать окно нужного размера оператором OPEN. Однако положение окна на экране, определяемого атрибутом AT(<ряд>,<колонка>), можно вычислять в процессе выполнения программы.

Размер окна не может быть больше размера экрана.

Техника работы со структурой SCREEN следующая. После открытия окна оператором OPEN старое содержимое экрана, "накрываемое" окном, сохраняется, и выводится новая информация. Затем в цикле выполняется оператор ACCEPT (получить значение поля), и в операторе CASE определяется, какое поле ввода было выбрано пользователем, и предпринимаются необходимые действия по обработке.

Очень похожим образом устроена и структура REPORT, описывающая отчет. За неимением места мы на ней останавливаться не будем.

Таблицы

Таблица является мощным средством языка, во многом компенсирующим отсутствие механизмов динамического выделения памяти под различные виды объектов.

Таблица представляет собой двухсвязный именованный список, размещаемый в памяти. Он состоит из элементов, аналогичных по структуре записям файла. Как и файл, таблица имеет переменный размер (в отличие от массива).

Работа с таблицей очень похожа на работу с файлом - используются те же операторы GET, PUT, DELETE, ADD, но работающие с таблицей. Существуют операторы работы с таблицой по номеру элемета таблицы и по ключу. Таблица может быть отсортирована в любом порядке командой SORT.

Таблица очищается командой FREE.

Описывается таблица структурным оператором TABLE.

Максимальное количество элементов в таблице 65536. Максимальный размер структуры TABLE 65520 байтов.

Дизайнер

Дизайнер - это программа, позволяющая производить проектирование прикладной программы методом "сверху вниз": создавать описания файлов данных, ключей и индексов, проектировать структуру программы и автоматически генерировать исходные тексты приложения. Дизайнер заменяет собой такие традиционные утилиты, как утилиту ведения проекта MAKE, генераторы меню, отчетов и экранов, и имеет некоторые дополнительные возможности.

Начинается разработка приложения с проектирования структуры файлов и их ключей, индексов и описания всех других необходимых атрибутов , а также глобальных данных. Эта работа проводится в специальном диалоговом окне дизайнера.

Затем создается структура программы: строится дерево модулей, именуются процедуры, присваивается им тип и описываются все их настраиваемые параметры.

В режиме диалога без программирования дизайнер позволяет создать четыре типа процедур:

- меню (Menu);

- таблицу (Table) (просмотр файла в виде таблицы и вызов процедур удаления, добавления и изменения. Ширина таблицы не может быть больше ширины экрана, и горизонтальный скролинг не осуществляется) ;
- экран (Form) (экранная форма для редактирования полей и переменных и отображения информации);
- отчет (Report).

Эти типы процедур имеют много настраиваемых параметров, задаваемых разработчиком в специальных диалоговых окнах дизайнера, что позволяет очень часто обойтись без их программирования вручную.

В дереве вызовов может присутствовать тип процедуры OTHER (другой). Это значит, что эта процедура создается не дизайнером, а кодируется программистом. Таким образом, в одном проекте возможно сочетать как автоматически генерируемые процедуры, так и написанные разработчиком.

После того как все это сделано, можно автоматически сгенерировать исходные тексты тех частей приложения, которые созданы дизайнером.

Вся информация о приложении содержится в файле описания приложения, который имеет расширение .APP. По окончании проектирования приложения дизайнер, используя так называемый модельный файл (имеющий расширение .MDL), генерирует исходные тексты процедур.

Модельный файл содержит макроопределения, по которым строится процедура каждого типа. Стандартно в составе пакета поставляются два модельных файла: для обычных и работающих в сети приложений. Пользователь может создавать свои модельные файлы, управляя таким образом генерацией исходного текста. Однако пользователь ограничен только типами процедур, поддерживаемыми дизайнером, так как они и их настраиваемые параметры жестко "зашиты" в программу Дизайнер.

Например, на рис. 1 и 2 представлены два основных окна дизайнера для приложения, работающего с одним файлом PERSON, и состоящего из двух модулей: просмотра файла в табличной форме и редактирования записи в экранной форме.

На рис. 3 показаны таблица и экранная форма, созданные с помощью дизайнера.

Тексты программы PERSON и процедуры PERSON_UPD приведены в листингах 1 и 2.

Размеры EXE модулей, полученных с помощью Clarion, редко бывают меньше 200-250K. Это связано с тем, что в состав программы включается большая по размеру исполняющая система, которая осуществляет обработку файлов, экранов и отчетов.

Вообще приложения, созданные с помощью дизайнера, получаются очень большими по объему, но на счастье хорошо поддающимися разрезке на оверлейные части.

Проблемы с использованием дизайнера возникают, когда такие действия, как удаление, добавление или изменение требуют модификации нескольких файлов нетривиальным образом. Например, переименование или удаление подразделения в файле подразделений предприятия, вероятно, потребует изменений в файле сотрудников и в файле балансов предприятия. Эта проблема решается двумя путями:

- кодированием таких процедур вручную. Для этого в настраиваемых параметрах автоматически генерируемых модулей предусмотрена возможность вставки в определенные места этих модулей вызовов процедур, созданных без помощи дизайнера;
- коррекцией модельного файла под конкретное приложение.

В одном из следующих номеров мы рассмотрим структуру модельного файла и покажем, как можно управлять генерацией исходного текста программы, меняя модельный файл.

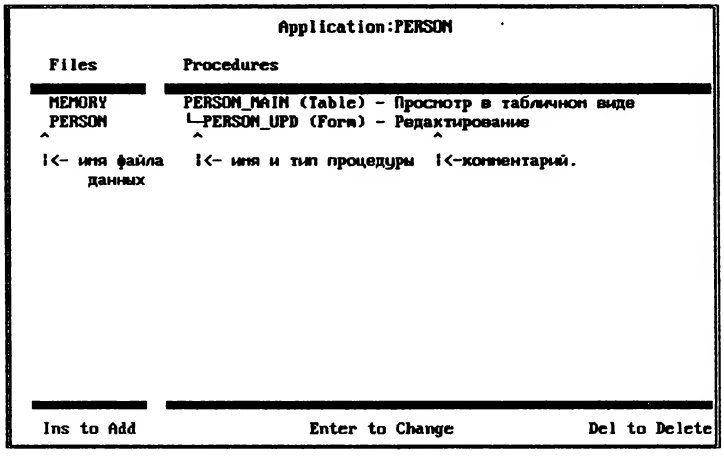


Рис. 1. Основное окно дизайнера

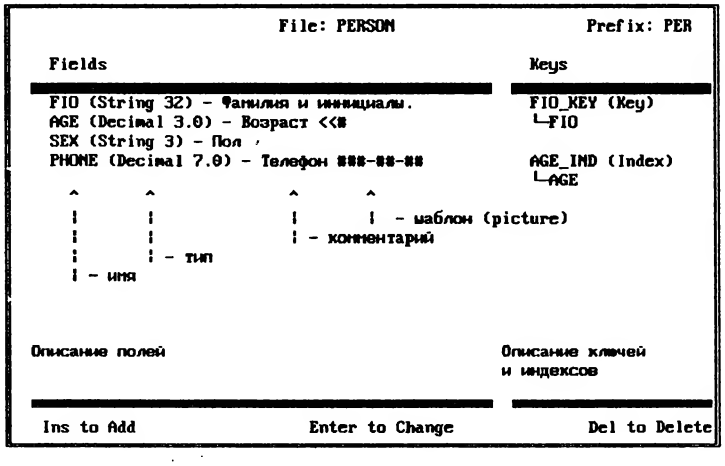


Рис. 2. Окно проектирования файла

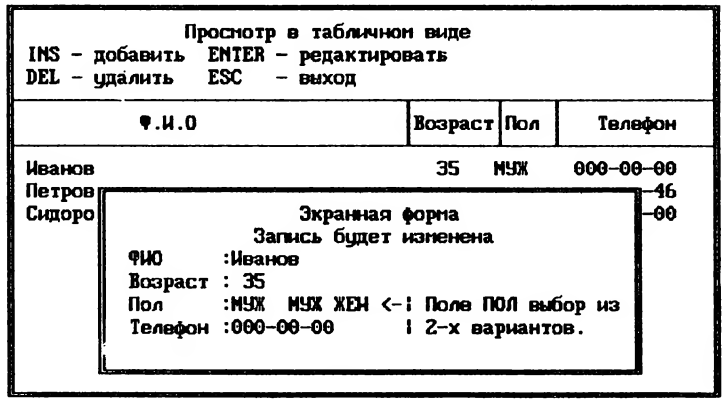


Рис. 3. Таблица и экранная форма, созданные с помощью дизайнера

Листинг 1. Файл PERSON.CLA:

```
PERSON      PROGRAM                    !Главная программа.
      !Вызывает процедуру PERSON_UPD для редактирования первой записи.
            INCLUDE('STD_KEYS.CLA')    !Включить определения
            INCLUDE('CTL_KEYS.CLA')    !кодов клавиш.
            INCLUDE('ALT_KEYS.CLA')    !Стандартные файлы
            INCLUDE('SHF_KEYS.CLA')    !определений.
REJECT_KEY  EQUATE(CTRL_ESC)           !Определить клавишу "ОТКАЗ"
ACCEPT_KEY  EQUATE(CTRL_ENTER)         !Определить клавишу "ПРИНЯТЬ"
TRUE        EQUATE(1)                  !Определить лог. "ИСТИНА"
FALSE       EQUATE(0)                  !Определить лог. "ЛОЖЬ"
            MAP                        !Описание структуры прогр..
              PROC(G_OPENFILES)        !В файле главной прогр..
              MODULE('PERSON1')        !Файл PERSON1.CLA содержит:
                PROC(PERSON_UPD)       !Процедуру работы с записью

PERSON      FILE,NAME('D:\MONITOR\PERSON'),PRE(PER),RECLAIM
FIO_KEY       KEY(PER:FIO),DUP,OPT     !Ключ.
AGE_IND       INDEX(PER:AGE),OPT       !Индекс.
RECORD        RECORD                   !Запись:
FIO             STRING(32)               !Фамилия и инициалы.
AGE           DECIMAL(3,0)             !Возраст
SEX           STRING(3)                !Пол
PHONE         DECIMAL(7,0)             !Телефон

      !Глобальные переменные - действие для UPD_PERSON.
ACTION      SHORT     !0 = нет действия;   1 = добавить;
                      !2 = редактировать;  3 = удалить.
      !Глобальная группа.
            GROUP,PRE(MEM)
MESSAGE     STRING(30)                 !Область сообщений.

      CODE                             !Кодовая часть.
      SETHUE(7,0)                      !Установить цвет:"Белый на черном"
      BLANK                            !Очистить экран
      G_OPENFILES                      !Вызвать процедуру открытия файла
      SETHUE()                         !Установить атрибуты цвета по умолчанию
      CLEAR(PER:RECORD)                !Очистить буфер записи
      SET(PER:FIO_KEY,PER:FIO_KEY)     !Установить первую запись по ФИО
      NEXT(PERSON)                     !Прочитать запись.
      ACTION = 2                       !Действие PERSON_UPD: редактирование
      PERSON_UPD                       !Вызов процедуры работы с записью.
      RETURN                           !Возврат в ДОС.
G_OPENFILES PROCEDURE                  !Открытие и проверка ошибок
      CODE                             !Кодовая часть.
      OPEN(PERSON)                     !Открыть файл.
      IF ERROR()                       !Если ошибка
        CASE ERRORCODE()               !Проверить
        OF 46                          !Неверный индекс
          BUILD(PERSON)                !Перестроить индекс
        ELSE                           !Вывести коды ошибок и закончить.
          LOOP; STOP('PERSON: ' & ERROR()).
```

Листинг 2. Файл PERSON1.CLA:

```
            MEMBER('PERSON')           !Главная программа PERSON.
      !Процедура осуществляет добавление, удаление, редактирование
      !записей файла PERSON.DAT. Эта процедура была сгенерирована
      !дизайнером. Комментарии переведены на русский язык и расширены.
PERSON_UPD  PROCEDURE                  !Процедура редактирования.
      ! Окно размером 9 на 48, цвет черный на сером.
SCREEN      SCREEN      PRE(SCR),WINDOW(9,48),HUE(8,7)
      !Описание текстов и рамок в окне:
              ROW(1,1)    STRING('╔{46}╗'),HUE(8,7)
              ROW(2,1)    REPEAT(7);STRING('║ <0{46}>║ '),HUE(8,7).
              ROW(9,1)    STRING('╚{46}╝'),HUE(8,7)
              ROW(2,19)   STRING('Экранная форма')
              ROW(4,4)    STRING('ФИО')
              ROW(5,4)    STRING('Возраст')
              ROW(6,4)    STRING('Пол')
              ROW(7,4)    STRING('Телефон')
              ROW(4,12)   STRING(':')
              ROW(5,12)   STRING(':')
              ROW(6,12)   STRING(':')
              ROW(7,12)   STRING(':')
      !Область для сообщения: добавление, удаление, редактирование.
MESSAGE     ROW(3,11)   STRING(30),ENH
                        ENTRY,USE(?FIRST_FIELD)
      !Поле ФИО: строка длиной 32, любые символы, верхний
      !регистр
              ROW(4,13)   ENTRY(@s32),USE(PER:FIO),LFT,UPR,REQ
      !Поле Возраст, число 3 цифры
              ROW(5,13)   ENTRY(@n3),USE(PER:AGE),NUM,REQ
      !Поле Пол, выбор из 2 пунктов, строка длиной 3
              ROW(6,13)   MENU(@s3),USE(PER:SEX),IMM,REQ
      !выбор из множества значений для поля пол.
                COL(18)     STRING('МУЖ')
                COL(22)     STRING('ЖЕН')

      !Поле Телефон, число 7 цифр. Заполнение поля необязательно.
              ROW(7,13)   ENTRY(@P###-##-##P),USE(PER:PHONE),NUM
                          ENTRY,USE(?LAST_FIELD)
      !Пауза для удаления.
                          PAUSE(''),USE(?DELETE_FIELD)

      CODE                     !Кодовая часть.
      OPEN(SCREEN)             !Открыть окно.
      SETCURSOR                !Выключить курсор.
      DISPLAY                  !Отобразить поля.
      LOOP                     !Бесконечный цикл.
        MEM:MESSAGE = CENTER(MEM:MESSAGE,SIZE(MEM:MESSAGE))
        DO CALCFIELDS          !Вычислить поля для показа.
        ALERT                  !Сбросить "горячие клавиши".
        ALERT(ACCEPT_KEY)      !Определить "гор." клавишу "ОТКАЗ".
        ALERT(REJECT_KEY)      !Определить "гор." клавишу "ПРИНЯТЬ".
        ACCEPT                 !Прочитать поле.
        IF KEYCODE() = REJECT_KEY THEN RETURN.   !Возврат по отказу.
        EXECUTE ACTION                           !Установить текст
          MEM:MESSAGE = 'Запись будет добавлена '  !в зависимости от
          MEM:MESSAGE = 'Запись будет изменена '   !значения
                                                   !глобальной
          MEM:MESSAGE = 'Для удаления нажмите ENTER '!перем. ACTION.

        IF KEYCODE() = ACCEPT_KEY  !По принятию:
          UPDATE                   !Переписать все поля с экрана.
```

```
    SELECT(?)                !Начать с текущего поля.
    SELECT                   !Редактировать.
    CYCLE                    !Перейти на начало цикла.

CASE FIELD()                 ! Перейти к редактированию полей.
OF ?FIRST_FIELD              ! Из первого поля выход по ESC
   IF KEYCODE() = ESC_KEY THEN RETURN.
   !Подтвердить удаление.
   IF ACTION = 3 THEN SELECT(?DELETE_FIELD).
OF ?PER:AGE                  !Возраст
   IF ~INRANGE(PER:AGE,0,999)  !Если значение не в диапазоне.
      BEEP                   ! Звуковой сигнал
      SELECT(?PER:AGE)       !и остаться на этом поле,
      CYCLE                  !и на начало цикла.

OF ?PER:PHONE                !Телефон
   IF ~INRANGE(PER:PHONE,0,9999999)  !Если значение
                             !не в диапазоне.
      BEEP                   ! Звуковой сигнал
      SELECT(?PER:PHONE)     !и остаться на этом поле,
      CYCLE                  !и на начало цикла.

OF ?LAST_FIELD               !Из последнего поля выход
                             !с изменением текущей записи и
   EXECUTE ACTION            !в зависимости от необходимого
                             !действия:
      ADD(PERSON)            !добавить,
      PUT(PERSON)            !изменить,
      DELETE(PERSON)         !удалить.

   IF ERRORCODE() = 40       !Ошибка: "Двойной ключ"
      MEM:MESSAGE = ERROR()  !Показать сообщение
      SELECT(2)              !в начале экр. формы
      CYCLE                  !выйти из цикла редактирования
   ELSIF ERROR()             !проверить на другие ошибки
                             !и, если
      STOP(ERROR())          !они есть, остановить выполнение

   ACTION = 0                !Установить признак "ВЫПОЛНЕНО"
   RETURN                    !Вернуться в вызывающую программу
OF ?DELETE_FIELD             !Если удаление
   IF KEYCODE() = ENTER_KEY  !и ENTER
   OR KEYCODE() = ACCEPT_KEY !или CTRL-ENTER
      SELECT(?LAST_FIELD)    !удалить запись,
   ELSE                      !иначе
      BEEP                   !сигнал и повторить еще раз.
...
CALCFIELDS      ROUTINE      !Локальная процедура
   IF FIELD() ?FIRST_FIELD   !Поле не первое ?
      IF KEYCODE() = 0 AND SELECTED() FIELD() THEN EXIT.!Выход.

   SCR:MESSAGE = MEM:MESSAGE !Вычислить вычисляемые поля.
```

ТРОЛЛЬ

Официальный представитель Horstmann Software Design Corporation

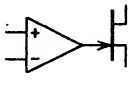
horstmann software design

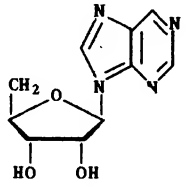
Редактор текста для научных целей

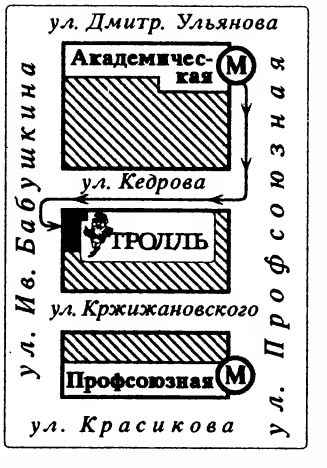
ChiWriter 4

Новая версия — новые возможности!

$$\sqrt{E\left(\frac{du}{dt}\right)^2 + 2F\frac{du}{dt}\frac{dv}{dt}}$$







Наш демо-центр: Москва, ул. Ив. Бабушкина, д. 11/2,
ИНОТЕХ-Прогресс: МП «Тролль». Тел.: (095) 124-9486

микроинформ

Локальные вычислительные сети

- Файловые серверы на базе IBM PC AT-386, IBM PC AT-486
- Сетевые жесткие диски от 300 Mb и выше
- Сетевые адаптеры и оборудование 3COM
- Платы дискового сопроцессора
- Коаксиальные и оптико-волоконные кабели
- Интеллектуальные устройства бесперебойного элетропитания

Программное обеспечение

- Операционные системы NetWare v2.2, v3.11
- Замена ОС NetWare на более поздние версии
- Средства работы с базами данных SQL, XQL, Btrieve
- Компилятор С для ОС NetWare
- Программы мониторинга и диагностики сети
- Программы сопряжения ОС NetWare с ОС UNIX

Авторизованный Учебный Центр фирмы

при СП МИКРОИНФОРМ

приглашает на курсы по изучению локальных вычислительных сетей

Инструкторы фирмы NOVELL проводят обучение
NetWare 386 v3.11 : System Manager Service & Support.

По окончании курса выдается сертификат фирмы NOVELL

Слушатели обеспечиваются учебными пособиями фирмы NOVELL на русском языке.

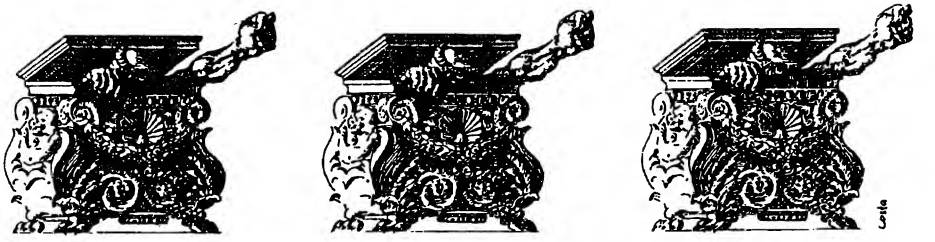
Ежегодно проводятся экзамены на степень Certified NetWare Engineer

Мы ждем Вас!

СП МИКРОИНФОРМ 113184 Москва ул. Островского д. 44 Коммерческий отдел
Учебный центр

Телефон : (095) 233 0006
(095) 235 7700

Телефакс: (095) 235 1053
Эл. почта: LEXICON@MICRO.MSK.SU.



Защищайтесь, сударь!

Часть 2

И.Василенко, Е.Гулаков

Управление базой дисков — средство для защиты и нападения

Работа с дисководом частично управляется особой таблицей, называемой базой дисков. Эта таблица при загрузке операционной системы помещается в оперативную память машины. База дисков имеет размер 11 байтов и содержит ряд параметров, которые определяют строение диска при его форматировании, а также характеристики работы дисковода.

Рассмотрим эти 11 байтов таблицы базы дисков.

0–й байт

В первых четырех битах содержится величина, показывающая время шага, то есть значение, соответствующее времени, выделенному дисководу для перемещения головок чтения/записи с одной дорожки на другую. Это время для IBM PC/AT составляет 6 миллисекунд. Во вторых четырех битах содержится величина, соответствующая времени вывода головок. Стандартное значение 0-го байта - DFh.

1–й байт

Если нулевой бит равен 1, то это указывает на включение режима прямого доступа к памяти (DMA). Следующие шесть битов указывают время ввода головок. Стандртное значение 1-го байта равно двум.

2–й байт

Это время ожидания отключения двигателя. То есть это промежуток времени, в течение которого двигатель остается еще включенным после завершения операции работы с дисководом. Обычно он составляет 55 миллисекунд. Это сделано для того, чтобы обеспечить ожидание повторных операций с НГМД без выключения мотора, которое ведет к потери времени на его разгон. Стандартное значение этого байта - 25h.

3–й байт

Этот байт содержит кодовое значение, соответствующее длине сектора. Значение этой величины следующее: 0 (128 байтов), 1 (256 байтов), 2 (512 байтов) или 3 (1024 байта). Изменение этого байта позволяет работать с секторами, размер которых отличается от 512- байтового формата MS DOS, принятого по умолчанию.

4–й байт

В нем находится номер последнего сектора на дорожке (для стандартной 360-Кбайтовой дискеты - это 9).

5–й байт

Этот байт задает величину промежутка для операций ввода/вывода. Его значение - 2Ah.

6–й байт

Максимальная длина сегмента данных, когда не задана длина сектора. Содержит стандартное значение - FFh.

7–й байт

Длина промежутка для операции форматирования. Стандартное значение - 50h.

8–й байт

Содержит значение, которое записывается при форматировании дискеты во все секторы. Его стандартное значение F6h. Изменение этой величины иногда применяется в целях защиты от копирования.

9–й байт

Это байт времени установки головки, то есть промежутка времени, за который происходит ее стабилизация после вывода в заданную точку дискеты. В 9-м байте содержится стандартное значение 19h.

10–й байт

Определяет время, отводимое для запуска двигателя. Стандартное значение этого байта - 2.

Из изложенного выше видно, что умение изменять параметры базы дисков является важным средством для создании систем защиты программ от несанкционированного копирования,так как изменение значений ее параметров влияют на работу процедур чтения/записи DOS. Указатель на базу дисков лежит в векторе прерываний 1Eh. Приведенная в листинге 1 программа BASA позволяет вызывать базу дисков и модифицировать ее параметры. Программа написана на языке Turbo C, однако может быть обработана любым другим C компилятором, который делает возможным опрос ячеек памяти по значениям OFFSET и SEGMENT. Все проведенные в базе дисков изменения оказывают влияние на последующие операции чтения/записи. Возвращение к таблице со стандартными значениями производится через перезагрузку машины или через изменение в программе значений параметров таблицы базы дисков на начальные.

В начале работы программа BASA присваивает элементам массива basa стандартные значения параметров таблицы базы дисков для работы с 360-Кбайтовыми дискетами. После этого вычисляются значения сегмента и смещения адреса, по которому располагается база дисков, используемая операционной системой MS-DOS в повседневной работе. Значения сегмента и смещения помещаются в массив u_oldb, а значения параметров таблицы стандартной базы дисков выводятся на экран. Затем можно изменить любые параметры и установить новую базу дисков. После экспериментов желательно сделать перезагрузку компьютера, прежде чем вернуться к работе в обычном режиме. Для опытов необходимо выбрать дискету, которая не хранит важные данные и программы, чтобы после измене-

ния базы дисков и попыток записи не утерять их навсегда. В остальном программа BASA не может нарушить правильной работы компьютера.

Оружейный арсенал в битве за сектор

Давайте разберемся в нескольких методах, которые будут составлять наш оружейный арсенал при написании программ защиты от копирования. Мы не ставим своей целью дать исчерпывающее описание этих методов. Успех созданной программы защиты целиком и полностью будет зависеть от фантазии изготовителя и нестандартности найденных решений, от того, насколько не знаком другим примененный в ней метод. Мы укажем только одно из направлений, идя по которому можно найти то, что сохранит Ваши секреты или будет оберегать Ваши права как автора.

Но прежде обратим внимание на следующее. Из-за возможных неудобств, связанных с работой программ защиты данных, некоторые пользователи от них просто отказываются. Поэтому необходимо подвергать свои программы всесторонней проверке, чтобы не затруднять работу рядового пользователя на компьютере.

Как Вы уже знаете, таблица базы дисков содержит важную информацию, необходимую для чтения, записи и форматирования дискет. Простейшая защита может заключаться в нестандартном форматировании отдельной дорожки, например, на 15 секторов вместо стандартных 9. Для этого необходимо лишь одно - найти возможность форматировать отдельную дорожку. Операционная система DOS предоставляет в распоряжение программиста механизм прерываний, который позволяет справиться с поставленной задачей.

Эта возможность демонстрируется программой SECT15, приведенной в листинге 2. Этой программой дорожка номер три разбивается процедурой форматирования на 15 секторов размером по 256 байтов каждый. После этого в первый сектор указанной дорожки заносится информация из массива secret. Чтобы произвести эти действия, необходимо установить базу дисков, которая позволяет заносить и извлекать информацию из секторов размером 256 байтов, имеющих номера от 1 до 15. Поэтому в начале программы устанавливается специальная база дисков, как было продемонстрировано в программе BASA. После завершения работ с нестандартными секторами, программа SECT15 восстанавливает исходную базу дисков. Информация, записанная в секторы третьей дорожки, не доступна для программ просмотра и копирования DOS. Известная NORTON утилита Disk editor, позволяя просматривать количество секторов на дорожке и их размер, также оставляет их содержание в тайне. Для извлечения информации из указанных секторов используется программа READS15, представленная в листинге 3.

Программа READS15, как и программа SECT15, вначале устанавливает нестандартную базу дисков, позволяющую читать секторы размером 256 байтов, имеющих номера от 1 до 15. Затем она считывает информацию из первого сектора третьей дорожки в массив secret и выводит ее на экран. В заключение программа READS15 восстанавливает стандартную базу дисков.

Часть процедур, обслуживающих дисководы, вызывается через прерывание 19 (13h). Это служебные процедуры базовой системы ввода/вывода. Номер вызываемой процедуры BIOS заносится в регистр AH. Для задач защиты от копирования наибольший интерес представляют следующие процедуры BIOS:

0 - сброс дисковой подсистемы в исходное состояние;
1 - чтение кода состояния дисковой подсистемы;
2 - запись секторов в память;
3 - чтение секторов из памяти;
4 - контроль секторов;
5 - форматирование дорожки.

Процедура с номером 0 устанавливает контроллер НГМД и дисковод в исходное состояние. Благодаря этой процедуре головки чтения/записи выводятся на нулевую дорожку, и все последующие операции с дискетами используют данное положение головок в качестве начала отсчета. Чаще всего эта процедура используется после различных сбоев и ошибок в работе дисковой подсистемы. Дополнительно в этой процедуре необходимо определить в регистре DL код дисковой подсистемы:

0 - соответствует дисководу A;
1 - соответствует дисководу B и т.д.

Процедура номер 1 записывает в регистр AL текущий статус дисковой подсистемы. Эта процедура характеризует состояние дисковой подсистемы после последней операции. Значения, соответствующие различным кодам статуса, можно найти, например, в таблице "Значения байта-статуса" широко известной справочной автоматизированной системы Norton Guide. Как и в процедуре номер 0, в регистр DL заносится код дисковой подсистемы.

Процедура номер 2 читает информацию из одного или нескольких секторов дискеты в память. Если необходимо прочитать несколько секторов, то все они должны лежать на одной дорожке и одной стороне дискеты, так как процедуры BIOS не могут в одно мгновение изменить активную дорожку и сторону дискеты. В регистрах необходимо определить следующие параметры.

AL - число читаемых секторов.

CH - номер дорожки, который обычно заключен между 0 и 39. (Замечание. Как отмечалось в первой части, иногда защита дискет строится таким образом, что дорожка 39 отформатирована так, что она не доступна для процедур чтения DOS,а дорожка 40 наоборот содержит четкую информацию, например, серийный номер или кодированные данные. В этом случае необходимо уметь читать дополнительную дорожку.)

CL - номер начального сектора при чтении.

ES:BX - содержит адрес области памяти, в которую будут записываться содержащиеся на диске данные. Зарезервированная область должна быть выбрана в соответствии с размером сектора, чтобы была возможность разместить в нее все данные. При этом необходимо заметить, что в обычном случае сектор занимает ровно 512 байтов, но может иметь длину вплоть до 1024 байтов.

CF - содержит код ошибки операции.

Процедура закончилась успешно, если CF=0. При установленном регистре флага (CF=1) в регистре AH располагается соответствующий код статуса дисковой подсистемы, который обозначает код ошибки, а в регистре AL возвращается число считанных секторов.

Процедура 3 (запись секторов), а также процедура 4 (проверка секторов) в своей работе используют те же регистры, что и описанная выше процедура 2. При проверке секторов можно отказаться от буфера данных,определяемого через регистры ES:BX, если не происходит записи или чтения данных. Процедура 4 проверяет, является ли сектор дефектным или нет, и правильно ли выводятся контрольные суммы.

Последняя процедура 5 осуществляет форматирование трека и является наиболее интересной для задач защиты от копирования. Каждый сектор дорожки имеет 4 байта, которые находятся в области данных и описывают этот сектор. Эти байты служат для опознания сектора. Они были подробно описаны в первой части, поэтому здесь мы только кратко напомним их назначение:

1-й байт - номер дорожки (цилиндра);
2-й байт - номер головки (0 или 1);
3-й байт - номер сектора;
4-й байт - код длины.

Чтение и запись сектора процедурами BIOS начинаются с поиска указанных 4 байтов, из которых байт, определяющий номер сектора, наиболее важный. Этот байт указывает, каким по счету сектор будет считан или записан при обработке секторов дорожки. Манипулируя номерами секторов можно увеличивать время доступа к сектору при последовательном чтении или даже добиться защиты информации, находящейся в определенном секторе. Рассматривая последовательность четырехбайтовых полей, мы извлекаем очень важные данные о строении дорожки. Для обычных 360-Кбайтовых дискет, стандартно отформатированных средствами DOS, данная последовательность, соответствующая дорожке 0 стороны 1, имеет вид :

0112 0122 0132 0142 0152 0162 0172 0182 0192

Для работы процедуры номер 5 необходимо в регистрах определить следующие значения:

AL - число секторов, создаваемых на дорожке;
CH - номер дорожки;
DH - номер головки;
DL - код дисковой подсистемы;
ES:BX - адрес области памяти, где находятся рассмотренные нами четырехбайтовые поля.

Ознакомившись с изложенными выше вопросами по форматированию дискет, можно смело приступать к экспериментам

с защитой данных. Через процедуру 5 прерывания 13h легко могут быть реализованы следующие элементы механизма защиты.

1.Можно легко поменять нумерацию секторов, что в определенных случаях оказывает влияние на время доступа к данным. Контролирующая программа отслеживает изменения в последовательности нумерации секторов или временные задержки и в зависимости от результатов тестирования разрешает или запрещает копирование.

2.Можно ввести несуществующий номер сектора, тогда программы чтения/записи при выходе на этот сектор будут выдавать ошибку. (См. программу METKA, представленную в листинге 4).

3.Один или более секторов дорожки могут быть отформатированы с различными длинами.

Теперь давайте разберемся, как механизм прерываний может быть реализован средствами языка программирования, и представим простую программу, использующую идеи этой статьи.

Programmer

Изменяя стандартные значения системной информации о параметрах магнитных дисков, можно получить эффективную систему защиты от копирования

Программирование вызова процедур BIOS и функций DOS на языке TURBO C

Доступ к специальным процедурам BIOS и функциям DOS, обслуживающим дисководы, осуществляется посредством вызова прерываний. Для функций DOS - прерывания номер 21h (десятичное значение 33), для процедур BIOS - прерывания номер 13h (десятичное значение 19).

В языке Turbo C есть несколько способов организации вызова прерываний. Рассмотрим наиболее общий, хотя в каждом конкретном случае можно выбрать и более рациональное решение. При этом способе происходит обращение к одной из двух функций int86 или int86x, прототип которых находится в файле dos.h. Общий вид функций следующий:

```
int86(int  intr_num,
  union  REGS  *inregs,
  union  REGS  *outregs)
int86x(int  intr_num,
  union  REGS  *inregs,
  union  REGS  *outregs,
  struct  SREGS  *segregs).
```

Здесь REGS - это объединение двух структур WORDREGS и BYTEREGS. Использование структуры типа WORDREGS позволяет рассматривать регистры центрального процессора как 16-битовые числа, BYTEREGS дает доступ к 8-битовым регистрам общего назначения AL,AH,BL,BH,DL,DH,CL,CH. Элементы структуры SREGS ставятся в соответствие сегментным регистрам. Обе эти функции выполняют программное прерывание, номер которого задается параметром intrnum. Перед выполнением программного прерывания обе функции копируют значения элементов объединения структур inregs в соответствующие им регистры. Кроме того, int86x копирует значения segregs.ds и segregs.es в регистры DS и ES, соответственно перед тем, как выполнить программное прерывание. Заметим, что inregs и outregs могут указывать на одно и то же объединение структур. Значения элементов структуры SREGS определяются либо путем присваивания, либо при помощи функции segread, которая присваивает элементам структуры значения сегментных регистров в момент вызова этой функции. Общий вид функции:

```
segread(struct  SREGS  *segregs)
```

Ее прототип находится в файле dos.h.

После выхода из программного прерывания обе функции помещают содержимое регистров в соответствующие элементы структуры outregs и копируют статус системы, сохраняя флаг состояния в outregs.x.cflag. Кроме того, int86x передает значение регистра DS. Если после завершения программного прерывания значение outregs.x.cflag не равно нулю, то это свидетельствует об ошибке.

В Turbo C существуют 6 различных моделей памяти. (Подробнее о них можно прочитать в документации по Turbo C).

В моделях памяти, называемых крошечной, малой и компактной, используются 16-битовые указатели на данные. Для соответствующих им данных они содержат смещение относительно фиксированного адреса сегмента. Значение этого сегмента загружено в регистр DS. Поэтому если в качестве буфера используется массив, то задача определения регистров ES:BX сводится к загрузке значения сегмента DS в ES, например, при помощи функции segread и структуры SREGS, и к загрузке в BX значения указателя, который является именем массива.

В моделях памяти, называемых средняя, большая и огромная, используются 32-битовые указатели на данные. Для работы с этим типом указателей существуют три функции :

- FP_SEG(far ptr) - устанавливает значение сегмента для указателя farptr, где farptr - указатель описанного типа;
- FP_OFF(farptr) - устанавливает смещение farptr;
- MK_FP(unsigned seg, unsigned off) - создает 32-битовый указатель из двух составляющих: сегмента seg и смещения off.

Используя эти функции, легко задать соответствующие значения элементам структур типа SREGS и WORDREGS, которые при вызове функции int86x будут определять значения пары регистров ES:BX.

Пример простой программы, защищающей авторские права

Теперь легко будет разобраться в работе двух простых программ, приведенных в листингах 4 и 5. Программы были написаны на языке Turbo C с использованием большой модели памяти. Эти программы помогут Вам идентифицировать дискеты незаметно для среднего программиста-пользователя. Причем, так как специальной обработке подвергается один из последних секторов дискеты, то ошибочные ситуации при работе средств DOS будут возникать крайне редко. Это помогает сохранять в тайне особый характер Ваших дистрибутивных дискет.

Программой, приведенной в листинге 4, Вы нестандартным образом переформатируете 39 дорожку и записываете в восьмой сектор выбранное Вами секретное слово, которое Вы определяете в массиве secret, используемом в качестве буфера. При попытке записать что-либо на эту дорожку средствами операционной системы DOS возникнет ошибка, и сектор восемь остается неизмененным.

При записи и чтении DOS отслеживает порядок нумерации секторов. После записи (чтения) информации сектора i будет производиться запись (чтение) сектора i+1, если этот сектор удовлетворяет параметрам, находящимся в базе дисков. Читая сектор семь, которому при форматировании был присвоен номер 11, DOS принимает решение, что все секторы дорожки исчерпаны. Таким образом, секторы 7,8,9 дорожки 39 остаются недоступными для программ чтения и записи DOS.

Размещая секретное слово на дискете при помощи процедуры прямой записи в сектор, мы не оставляем информации о нем в управляющих блоках нашей дискеты, поэтому выйти на эту запись можно, только зная ее физический адрес или имея возможность просматривать любой участок дискеты. Такие возможности предоставляют некоторые NORTON утилиты, но время, затраченное на поиски такого закрытого участка, в общем случае может оказаться достаточно большим. Ни один из имеющихся на дискете каталогов не будет содержать информации о месте расположения нашего секретного слова.

Программа EXAM осуществляет чтение заданного буфера дискеты и сравнение соответствующей его информации с "маркантом" дистрибутивной дискеты. При этом используется функция biosdisk, которая служит непосредственно для выполнения операций с диском в BIOS. В принципе описанные ранее операции, вызываемые через прерывание 13h, могут быть реализованы этой функцией. При этом в рассмотренных операциях BIOS значения регистра AH и значения параметра cmd в функции biosdisk при совпадении указывают на одну и ту же операцию. В силу простоты работы с функцией biosdisk мы ее в данной статье не рассматриваем.

Листинг 1

```
/*                          Программа BASA.C */
#include <dos.h>
#include <stdio.h>
main()
{
     struct WORDREGS{
          unsigned int ax,bx,cx,dx,si,di,cflag,flags;
               };
          struct BYTEREGS{
               unsigned char al,ah,bl,bh,cl,ch,dl,dh;
               };
          struct SREGS{
               unsigned int es,cs,ss,ds;
               };
          union REGS {
               struct WORDREGS x; struct BYTEREGS h;
               };
          union REGS in,out;
          struct SREGS segr;
          char *ptr;
     char basa[11];      /*   база дисков    */

     char bold,an;
     int u_oldb[2],i,j,k;
/*_____________________________________________________________*/
     basa[0] = 0xDF; /* установка стандартной базы дисков */
     basa[1] = 2; /* для 360-Кбайтовых дискет */
     basa[2] = 0x25; basa[3] = 2; basa[4] = 9; basa[5] = 0x2A; basa[6] = 0xFF;
     basa[7] = 0x50; basa[8] = 0xF6; basa[9] = 0x19; basa[10] = 2;
/*_____________________________________________________________*/
     in.h.al = 0x1E;
     in.h.ah = 0x35;
     segread(&segr);              /* считывание значений сегм.регистр */
     int86x(33,&in,&out,&segr);  /* получение вектора прерывания 1Eh */
     u_oldb[0] = out.x.bx;
     u_oldb[1] = segr.es;
     ptr = MK_FP(segr.es,out.x.bx);/* создание указателя на старую */
        printf("\n");             /* базу дисков */
        for (i = 0;i!+ + ) {
        printf("Параметр ");
        printf("%d",i);           /* распечатка текущей базы дисков */
        printf(" : ");
        bold = *(ptr+i);
        printf("%x\n",bold);
          }
/*_____________________________________________________________*/
     printf("Хотите изменить базу дисков ? ");
     printf("Ответьте y или n: ");
     scanf("%1c",&an);
     fflush(stdin);
     if (an = ='y'){
          for(i = 0;i!+ + ){
          printf("Хотите изменить параметр ? ");
          printf("%d\n",i);
          printf("Ответьте y или n : ");
          scanf("%1c",&an);
          fflush(stdin);
          if (an = ='y'){
               printf("Введите новый параметр : ");
               scanf("%x",&k); /* ввод hex значения i-го */
               fflush(stdin); /* параметра */
               basa[i] = k;
              }
     in.h.al = 0x1E;
     in.h.ah = 0x25;
     in.x.dx = FP_OFF(basa);       /* установка новой базы дисков */
     gena.ds = FP_SEG(basa);
int86x(33,&in,&out,&segr);
     }
/*_____________________________________________________________*/
     else {
     in.x.dx = u_oldb[0];
     gena.ds = u_oldb[1];
     in.h.al = 0x1E;       /* восстановление базы дисков, */
     in.h.ah = 0x25;       /* установленной перед запуском программы */
     int86x(33,&in,&out,&segr);
          }
     printf("**** Программа завершена ***");
}
```

Листинг 2

```
/*                          Программа SECT15.C */
#include <dos.h>
#include <stdio.h>
main()
{
struct WORDREGS{
unsigned int ax,bx,cx,dx,si,di,cflag,flags;
};
     struct BYTEREGS{
          unsigned char al,ah,bl,bh,cl,ch,dl,dh;
               };
     struct SREGS{
          unsigned int es,cs,ss,ds;
               };
     union REGS {
          struct WORDREGS x; struct BYTEREGS h;
               };
     union REGS in,out;
     struct SREGS segr;
     char basa[11];      /*   база дисков    */
     int u_oldb[2];
     char addr[60];      /*   буфер четырехбайтовых полей */
     char secret[512];   /*   буфер для записи */
     int i,j;
     /*_____________________________________________________*/
     secret[0] = 't'; secret[1] = 'a'; secret[2] = 'й'; secret[3] = 'н';
     secret[4] = 'о';
     secret[5] = 'е';       /*   подготовка скрытой информации */
     secret[6] = ' '; secret[7] = 'с'; secret[8] = 'л'; secret[9] = 'о';
     secret[10] = 'в'; secret[11] = 'о'; secret[12] = ' ';
     for (i = 13;i < 512;i+ + )
          secret[i] = 0;
     j = 1;
     for (i = 0;i < 60;i = i+4) {
          addr[i] = 3;
          addr[i+1] = 0;       /*   определение четырехбайтовых полей */
          addr[i+2] = j; addr[i+3] = 1;
          j = j+1;
               }
     basa[0] = 0xDF; basa[1] = 2; basa[2] = 0x25; basa[3] = 2;
     basa[4] = 9; /* задание специальной базы дисков */
     basa[5] = 0x2A; basa[6] = 0xFF; basa[7] = 0x50; basa[8] = 0xF6;
     basa[9] = 0x19; basa[10] = 2;
/*_____________________________________________________*/
     in.h.al = 0x1E; in.h.ah = 0x35;
     segread(&segr);
     int86x(33,&in,&out,&segr); /*     получение вектора прерывания 1Eh */
     u_oldb[0] = out.x.bx;
     u_oldb[1] = segr.es;
/*_____________________________________________________*/
     segread(&segr);
     gena.es = FP_SEG(addr);
     in.x.bx = FP_OFF(addr); in.x.dx = 0;
     in.h.ch = 3;          /*   нестандартное форматирование 3-й дорожки */
     in.h.ah = 0x05; in.h.al = 15;
     int86x(19,&in,&out,&segr);
/*_____________________________________________________*/
     in.h.al = 0x1E; in.h.ah = 0x25;
     in.x.dx = FP_OFF(basa); /*   установка специальной базы дисков */
     gena.ds = FP_SEG(basa);
     int86x(33,&in,&out,&segr);
/*_____________________________________________________*/
     segread(&segr);
     gena.es = FP_SEG(secret);
     in.x.bx = FP_OFF(secret);
     in.x.dx = 0;
     in.h.ch = 3; in.h.cl = 8;
     in.h.al = 2;          /*   запись данных в секторы нестан- */
     in.h.ah = 3;          /*   дартной дорожки */
     int86x(19,&in,&out,&segr);
/*_____________________________________________________*/
     in.x.dx = u_oldb[0];
     gena.ds = u_oldb[1];
     in.h.al = 0x1E; /*   восстановление исходной базы дисков */
     in.h.ah = 0x25;
     int86x(33,&in,&out,&segr);

     printf("normall");

}
```

Листинг 3

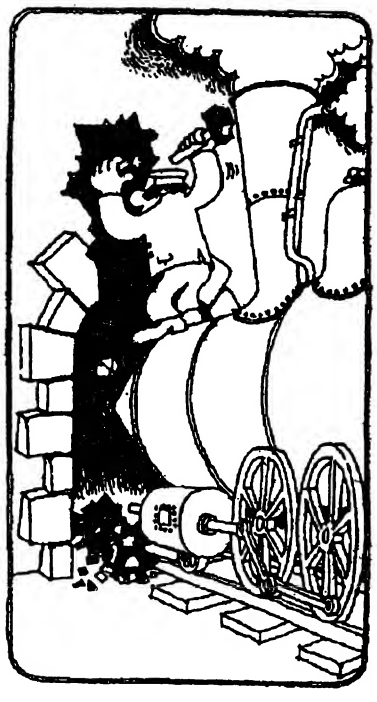
```
/*                    Программа READS15.C */
#include <dos.h>
#include <stdio.h>
main()
{
    struct WORDREGS{
        unsigned int ax,bx,cx,dx,si,di,cflag,flags;
        };
    struct BYTEREGS{
        unsigned char al,ah,bl,bh,cl,ch,dl,dh;
        };
    struct SREGS{
        unsigned int es,cs,ss,ds;
        };
    union REGS {
        struct WORDREGS x; struct BYTEREGS h;
        };
    union REGS in,out;
    struct SREGS segr;
    char basa[11];      /*  база дисков          */
    int u_oldb[2];
    char addr[60];
    char secret[1024]; /*  буфер данных         */
    int i,j;
/*-------------------------------------------------*/
    for (i=0;i<1024;i++)
        secret[i] = 0;
    j=1;
    for (i=0;i<60;i=i+4){
        addr[i] = 3;    /*  определение четырехбайтовых полей */
        addr[i+1] = 0; addr[i+2] =j; addr[i+3] = 1;
        j=j+1;
          }
    basa[0] = 0xDF; basa[1] = 2;
    basa[2] = 0x25; /* создание специальной базы дисков */
    basa[3] = 1; basa[4] = 15; basa[5] = 0x2A; basa[6] = 0xFF;
    basa[7] = 0x50; basa[8] = 0xF6; basa[9] = 0x19; basa[10] = 2;
/*-------------------------------------------------*/
    in.h.al = 0x1E; in.h.ah = 0x35;
    segread(&segr);
    int86x(33,&in,&out,&segr);  /*  получение вектора прерывания 1Eh */
    u_oldb[0] = out.x.bx;       /*  (адреса старой базы дисков) */
    u_oldb[1] = segr.es;
/*-------------------------------------------------*/
    in.h.al = 0x1E; in.h.ah = 0x25;
    in.x.dx = FP_OFF(basa);     /*  установка специальной базы дисков */
    gena.ds = FP_SEG(basa);
    int86x(33,&in,&out,&segr);
/*-------------------------------------------------*/
    segread(&segr);
    gena.es = FP_SEG(secret);
    in.x.bx = FP_OFF(secret);
    in.x.dx = 0;
    in.h.ch = 3; in.h.cl = 1;
    in.h.al = 1;                /*  чтение секторов особой дорожки */
    in.h.ah = 2;
    int86x(19,&in,&out,&segr);
    for(i=0;i<14;putchar(secret[i]),i++);
/*-------------------------------------------------*/
    in.x.dx = u_oldb[0];
    gena.ds = u_oldb[1];
    in.h.al = 0x1E; in.h.ah = 0x25;
    int86x(33,&in,&out,&segr);  /*  установка исходной базы дисков */
    printf("*** Все нормально ***");
}
```

Листинг 4

```
/*                    Программа METKA.C */
#include <dos.h>
#include <stdio.h>
main()
{
    struct WORDREGS{
        unsigned int ax,bx,cx,dx,si,di,cflag,flags;
        };
    struct BYTEREGS{
        unsigned char al,ah,bl,bh,cl,ch,dl,dh;
        };
    struct SREGS{
        unsigned int es,cs,ss,ds;
        };
    union REGS {
    struct WORDREGS x; struct BYTEREGS h;
        };
    union REGS in,out;
    struct SREGS segr;
    char addr[36]; char secret[512];
    int i,j;
/*-------------------------------------------------*/
    secret[0] = 'c'; secret[1] = 'o'; secret[2] = 'n'; secret[3] = '.';
    secret[4] = ' '; secret[5] = 'c'; secret[6] = 'e'; secret[7] = 'k';
    secret[8] = 'p'; secret[9] = 'e'; secret[10] = 'r'; secret[11] = 'n';
    secret[12] = 'o';
    for (i=13;i<512;i++)
        secret[i] = 0;
    j=1;
    for (i=0;i<36;i=i+4){
        addr[i] = 39; addr[i+1] = 0; addr[i+2] =j; addr[i+3] = 2;
        j=j+1;
          }
    addr[26] = 11;
/*-------------------------------------------------*/
    segread(&segr);

    gena.es = FP_SEG(addr);
    in.x.bx = FP_OFF(addr);
    in.x.dx = 0;
    in.h.ch = 39;          /*  нестанфартное форматирование */
    in.h.ah = 0x05;        /*  39-й дорожки */
    in.h.al = 9;
    int86x(19,&in,&out,&segr);
/*-------------------------------------------------*/
    segread(&segr);
    gena.es = FP_SEG(secret);
    in.x.bx = FP_OFF(secret);
    in.x.dx = 0; in.h.ch = 39; in.h.cl = 8;
    in.h.al = 1; /* запись */
    in.h.ah = 3;
    int86x(19,&in,&out,&segr);
/*-------------------------------------------------*/
    printf("*** Программа завершилась успешно ***");
```

Листинг 5

```
/*                    Программа EXAM.C*/
#include <bios.h>
#include <stdio.h>
main()
{
    int cmd,drive,head,track,sector,nsects,i,j;
    char m1,m2,buffer[512],secret[13];
/*-------------------------------------------------*/
    secret[0] = 'c'; secret[1] = 'o'; secret[2] = 'n'; secret[3] = '.';
    secret[4] = ' '; secret[5] = 'c'; secret[6] = 'e'; secret[7] = 'k';
    secret[8] = 'p'; secret[9] = 'e'; secret[10] = 'r'; secret[11] = 'n';
    secret[12] = 'o';
    cmd = 2;
    drive = 0;
    head = 0;
    track = 39;
    sector = 8;
    nsects = 1;
    for(i=0;i<3;i++)
    biosdisk(cmd,drive,head,track,sector,nsects,buffer);
    j=0;
    for (i=0;i<13;i++){
        m1 = buffer[i];
        m2 = secret[i];
    if (m1 == m2)
        ;
    else j = 1;
        }
    if (j! = 0)
        printf("Ай-ай-ай, но это пиратская копия !!!");
    else
        printf("Это разрешенная копия");
}
```



Дороги, которые мы выбираем

От редакции

Начиная с этого номера журнала, мы будем публиковать в рубрике "Игры" статьи, содержащие экспертные проработки некоторых видов игрового программного обеспечения. Статья "Дороги, которые мы выбираем" - первая из серии таких статей-экспертиз, подготовленных научно-технической фирмой "ИНФОРКОМ", которая специализируется на разработке обучающих программ для персональных компьютеров типа IBM PC XT/AT.

Вступление

В наших средствах массовой информации в определенной мере сложилось негативное отношение к игровой тематике. Во многом это вызвано подходом к этому жанру, как к несерьезному, а также беспредельным несанкционированным копированием игровых программ, что грубо нарушает авторские права фирм-разработчиков.

Все это действительно так, но нельзя отрицать того гигантского вклада, который внесло игровое программное обеспечение в развитие программирования как искусства. Здесь есть, что позаимствовать очень многим профессионалам. Идеология интерфейса человек-компьютер, динамика управления экраном, создание атмосферы аддиктивности и духа конкуренции – вот в первую очередь те черты, которые могут быть приняты на вооружение при разработке более серьезных, в частности обучающих программ и систем.

Активно работая более трех лет над оригинальным обучающим программным обеспечением, "ИНФОРКОМ" внимательно следит за интересными игровыми программами. Лучшая обучающая программа - всегда в какой-то степени игра, а игровая программа никогда не достигнет коммерческого успеха, если она ничему не учит и ничего не развивает.

Открывая эту серию статей на страницах журнала "Монитор", мы сразу предупредим читателя, что мы не собираемся ни печатать инструктивные материалы, помогая пиратам нарушать авторские права, ни делать обзоры рекламного характера. Наша цель - в том, чтобы установить диалог с читателями, привить вкус к исследовательской работе, разработке дебютных стратегий, анализу вариантов.

Сегодня Вашему вниманию предлагается первая попытка анализа такой программы. Мы и далее будем рассматривать только те программы, которые требуют специальных знаний и могут служить для повышения общего уровня образованности и информированности. Это прежде всего имитаторы сложных технических устройств, стратегические игры, игры делового плана и игры жанра ADVENTURE, являющиеся неплохим пособием по английскому языку для нашего пользователя.

Мы ждем Ваших писем с идеями, предложениями, пожеланиями. Прежде всего мы хотели бы получить Ваши предложения по дебютной стратегии для высшего уровня сложности рассмотренной ниже программы RAILROAD TYCOON.

Ваши данные явятся основой для дальнейшего анализа, лучшие и наиболее оригинальные идеи мы надеемся опубликовать. Рекомендуем взять за основу анализа создание транспортной сети в Европе.

Программа: RAILROAD TYCOON
Жанр: SIMULATION/MANAGEMENT
Фирма: MICROPROSE
Автор: Sid Meier
Год выпуска: 1990
Дисплей: CGA/EGA/MCGA/VGA

Общая характеристика

Программные имитаторы фирмы MICROPROSE широко известны отечественному пользователю. Достаточно упомянуть имитатор истребителя-бомбардировщика F15 "Eagle" и имитатор крейсерской подводной лодки типа "K" - "Silent Service". Не отличаясь особой плодовитостью по количеству выпускаемых программ, фирма уделяет особое внимание их качеству. Программы отлично сбалансированы, имеют дружественный пользовательский интерфейс и оставляют неизгладимое впечатление от эффекта реального присутствия.

Программа "Railroad Tycoon" является имитатором железной дороги. По итогам 1990 года она была признана лучшим имитатором года, и это не случайно. Строго говоря, ее квалификация как имитатора (SIMULATION) является более чем условной и отражает только тот отрадный факт, что авторам удалось добиться высокого уровня соответствия модели и реальной действительности. На самом же деле найдется совсем немного программ, превосходящих ее в жанре деловой игры (MANAGEMENT). В программе очень удачно сочетаются обе генеральные ветви современного менеджмента - менеджмент практический (оперативное управление) и менеджмент стратегический (стратегическое планирование).

Программа имеет серьезный обучающий и развивающий эффект. Она дает элементарное представление о сущности и принципах деятельности фондового рынка (рынка ценных бумаг), является неплохим пособием по экономической географии Европы и США, развивает навыки оперативного управления сложным промышленным объектом. По-видимому, именно возможность попробовать свои силы в самых разнообразных профессиях в рамках одной игры и является тем фактором, который обеспечил программе коммерческий успех.

Сюжет игры

Вам предоставлена возможность попробовать свои силы в качестве президента железнодорожной компании. Для своей деятельности Вы можете выбрать один из четырех экономически развитых промышленных регионов:

Англия, 1828 г.
Атлантическое побережье США, 1830 г.
Тихоокеанское побережье США, 1866 г.
Европа, 1900 г.

Выбор региона имеет значение в том смысле, что стратегия Вашей деятельности при переходе от региона к региону может меняться столь значительно, что Вы имеете как бы четыре разных программы в одной.

Свою предпринимательскую карьеру Вы начинаете с одним миллионом фунтов стерлингов (в США - долларов). Они собираются путем распространения акций компании среди населения. Вы распространили акции двух типов - простые (SHARES) и привилегированные (BONDS). Между ними есть разница, и ее надо знать.

Привилегированные акции на сумму 500 тыс. ф.ст. дают право Вашим кредиторам на получение стабильного процента от суммы вклада независимо от прибыльности компании, т.е. для Вас это как бы долговые обязательства, процент по которым Вы платите всегда. Но здесь есть и преимущество. Привилегированные акции не дают их владельцам права голоса на общем собрании акционеров.

Простые акции в количестве 100 тыс. по цене 10 ф.ст. за штуку Вам удалось распространить тоже за 500 тыс. ф.ст. По этим акциям прибыль распределяется в виде дивидендов и, если очередной двухлетний финансово-отчетный период Вы закончили с убытком, можете ничего акционерам не платить, правда при этом курс Ваших акций на фондовой бирже неизбежно понизится. Держатели простых акций (SHAREHOLDERS) вполне могут в случае неудовлетворенности показателями отозвать Вас с поста президента и на этом Ваша предпринимательская карьера будет закончена.

Итак, Вы приняли решение о строительстве железной дороги и далее успех Вашего предприятия целиком зависит от оптимальности целой серии практических шагов:

– конкретного выбора территории под строительство трассы (F2, F3, F4);
– определения оптимальной структуры транспортной сети (BUILD TRACK);
– экономически обоснованного выбора станций (STATIONS);
– создания инфраструктуры транспортного хозяйства (IMPROVE STATION);
– установления оптимального количества составов;
– выбора локомотивов (ENGINE) и составов (TRAIN);
– анализа схемы грузопассажирских потоков, разработки маршрута движения поездов и расписания остановок (SHEDULE). Принятые на этом этапе стратегические решения будут положены в основу концепции будущей железнодорожной сети. Эти решения должны быть согласованы с экономической характеристикой региона, с наличием ресурсов, с оценкой потребности региона в сырье, материалах и кадрах.

Дальнейшая деятельность строится как оперативное управление железнодорожным хозяйством. На всех этапах работы системы Вы можете отдать приказ на изменение маршрута или расписания остановок любого поезда. Вы можете задать режим формирования состава на любой из промежуточных станций и оперативно вносить необходимые коррективы. Основным критерием правильности принятия тех или иных решений является наличие прибыли по итогам отчетного финансового периода и ее размер.

Ваши действия ограничены размерами свободного финансового капитала, который может быть подкреплен операциями на бирже:

– продажей акций своей компании или компаний конкурентов (SELL STOCK);

– приобретением акций (BUY STOCK);

– выпуском или обратным выкупом долговых обязательств (CASH).

Конкурентная борьба

Одним из интереснейших аспектов программы, вносящим захватывающую динамику и регулирующим игровой баланс, является конкурентная борьба. Острота конкурентной схватки меняется в зависимости от уровня сложности. На низшем уровне (INVESTOR) Вы можете об этом почти не думать и сосредоточиться на оптимальном развитии своей компании. На высшем же уровне (TYCOON) при максимальном приближении к реальности (CUT THROAT COMPETITION) Вам приходится внимательно следить за своими конкурентами. Наиболее простой и пассивный прием борьбы – окружение. Суть его в том, что конкурирующие компании застраивают своими транспортными сетями регионы в непосредственной близости от Вашего. Поскольку Вы не имеете права подключаться к чужой транспортной сети, возможности Вашего дальнейшего развития оказываются проблематичными. Такой прием характерен для низших уровней сложности, т.к. деятельность компаний на высшем уровне протекает достаточно трудно и конкурентам непросто создать громоздкие сети для локализации Вашего операционного поля.

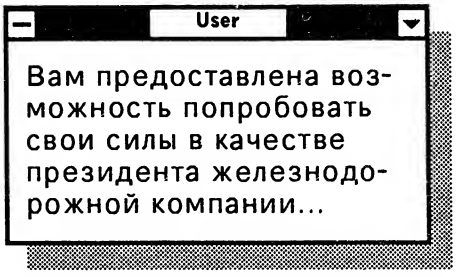
На высшем уровне сложности огромную роль приобретает биржевая игра. При наличии достаточного свободного капитала (а за ним Вы должны следить) конкуренты скупают Ваши акции и здесь атака носит активный характер и имеет два нежелательных для Вас последствия:

а) скупив более 50% Ваших акций, конкуренты организуют собрание акционеров и меняют руководство, а попросту говоря - увольняют Вас с должности президента;

б) сдерживание биржевой атаки конкурента вынуждает Вас скупать акции собственной компании. Каждая покупка акций сопровождается курсовой надбавкой, таким образом, Вы скупаете акции по искусственно завышенным ценам, экономя на развитии производства. Поскольку в этот период завышенный курс акций не соответствует реальной прибыльности Вашей компании, то их цена неизбежно упадет в новом отчетно-финансовом периоде и все потери лягут на Ваши плечи, ухудшая и без того трудное материальное положение.

Естественный прием самозащиты - своевременное приобретение акций собственной компании в начале игры (TREASURE STOCK) в количестве, превышающем их наличие на руках у основного конкурента, но это конечно связывает оборотный капитал и мешает нормальному развитию фирмы.

Наиболее трудный вид конкурентной борьбы - тарифная война (RATE WAR). Она начинается, когда конкуренты включают Вашу станцию в свою транспортную сеть. В этом случае город, в котором произошло столкновение интересов, становится полем битвы. Неуклонное снижение тарифов на услуги по доставке грузов и пассажиров ведет к многократному снижению доходов в этом пункте, и Ваша деятельность принимает убыточный характер с последующим резким снижением курса Ваших акций.



Падение курса Ваших акций делает Вас не только легкой добычей конкурентов. Она влечет и естественное неудовольствие Ваших акционеров, и если Вы не владеете 50% (контрольным пакетом) акций собственной компании, акционеры попросят Вас освободить пост президента.

Тарифная война сопровождается и политической борьбой за влияние на членов городского муниципалитета. Если свои услуги по доставке и вывозу грузов Вы будете исполнять хуже, чем конкурент, Вам предложат покинуть город навсегда с потерей имущества, составов и т.п. Так может быть разрушена структура Вашей сети, и как правило это предвещает близкий конец карьеры.

Побеждать в тарифной войне можно только быстро и решительно. Победа достигается, если объем Ваших услуг не менее чем в два раза превысит объем услуг конкурентов. Затяжной характер борьбы ведет к разорению за счет падения доходов.

Ведя тарифную войну, надо безусловно обеспечить быструю оборачиваемость поездов и наиболее полный вывоз грузов. Количественные показатели здесь не единственные. Необходимо сделать все возможное для выполнения услуг и по номенклатуре товаров как ввозимых, так и вывозимых.

Особенности экономических регионов

Наличие четырех различных промышленно развитых регионов в игре вносит приятное разнообразие, особенно на высшем уровне сложности. Как правило характер развития Вашей фирмы соответствует социально-экономической характеристике региона в рассматриваемый период деятельности.

Англия, 1928г.

Этот регион характеризуется очень низкой нормой прибыли от транспортных операций и незначительными грузопотоками. Приходится бороться за каждую тонну груза, за всемерное снижение эксплуатационных расходов. В то же время биржевая игра не имеет значительного удельного веса. Развитие конкурентов - замедленное, и тарифная война наступает не скоро. Основная цель - экономическая эффективность операций. Каждый период должен заканчиваться пусть с незначительной, но прибылью. Выплата процентов по займам - невыносима. Привилегированные акции (BONDS) следует выкупать в самом начале развития. Инвестиции в промышленность (BUILD INDUSTRY) как правило неэффективны - следует всемерно использовать имеющиеся ресурсы.

Наиболее эффективны операции по перевозке скота (LIVESTOCK), угля (COAL), пива (BEAR) и сырья для его производства (CHOPS).

Рельеф местности следует выбирать наиболее простой (локомотивы не имеют достаточной мощности).

Высок удельный вес специальных экспрессных доставок (PRIORITY DELIVERY). Доходы от них сравнимы с доходами от основной деятельности и могут даже превосходить.

Атлантическое побережье США, 1830 г.

Бурный экономический рост этого региона обеспечивает наличие высокой прибыльности транспортных операций. Достаточно высокая плотность населенных пунктов создает потенциал для хорошего развития сложной структуры с многочисленными составами.

Конкурентная борьба рано приходит к своим острейшим стадиям, так как Ваше бурное развитие сопровождается и быстрым развитием конкурентов. Основной вид борьбы - тарифная война. Игра имеет динамичный, остротекущий характер.

Простой рельеф местности также обеспечивает динамичность операций и остроту конкурентной борьбы.

Тихоокеанское побережье США, 1866 г.

Регион также характеризуется высокой нормой прибыли. В то же время значительные масштабы обслуживаемых территорий позволяют выделить основную массу доходов на развитие транспортной сети, в малой степени опасаясь атак конкурентов на свои пункты. Сложный рельеф местности требует значительных инвестиций, но может служить неплохим прикрытием от прямых атак соперников.

Конкурентная борьба развивается в основном на фондовой бирже. Если вам удалось защититься от биржевых атак конкурентов, то имеет смысл самому попробовать сделать биржу источником дохода. Угадать, акции каких компаний будут иметь рост в ближайший период - вот главная задача в этой борьбе, требующая определенного опыта. Внимательно следите за всеми газетными сообщениями.

Европа, 1900 г.

Это наиболее сложный регион, сочетающий в себе необходимость считать каждый фунт, как в Англии, и динамично управлять ситуацией, как в Америке.

Очень многое зависит от правильности выбора опорного пункта будущей сети. На карте Европы при высокой плотности населения, оказывается совсем немного пунктов, способных привести к успеху.

Основным видом деятельности являются почтово-пассажирские перевозки. Развивая промышленность, можно обеспечить наличие и других грузов.

Значительный удельный вес имеет диспетчерская работа по управлению формированием и движением составов.

Практические рекомендации

1. Выбирая объект для начала строительства железнодорожной сети, внимательно проанализируйте распределение ресурсов по региону. Карту распределения ресурсов можно визуализировать, нажимая на клавиши F2 или F3, если предварительно был включен режим RESOURCE MAP.

2. Обычно начинающие менеджеры сталкиваются с трудностями при прокладке своей первой железнодорожной колеи. Они связаны с тем, что в программе реализован несколько нестандартный способ управления. Находясь в режиме BUILD TRACK, нажмите клавишу SHIFT и стройте путь, управляя клавишами дополнительной (цифровой клавиатуры).

3. Прокладывая путь, не забывайте сверяться с результатами топографических изысканий. Сведения о рельефе местности можно получить, нажимая на клавишу F10.

4. Внимательно следите за газетными сообщениями о действиях Ваших конкурентов. Они не только предупредят Вас о начале биржевой атаки, но могут и дать ключ для того, чтобы угадать, акции какой компании пойдут вверх. Одним из верных признаков является, например, сообщение о том, что некая компания, имеющая очень низкий курс акций, проложила новую трассу.

5. Бывают случаи, когда включение очередного города в Вашу транспортную сеть вызывает просьбу населения этого города о выпуске и продаже дополнительного пакета Ваших акций. Если Вы планируете и дальнейшее развитие своей сети, не надо им в этом отказывать. Их просьба свидетельствует о доверии к Вашей компании и дает Вам живые деньги.

6. Очень аккуратно относитесь к долговым обязательствам. На высшем уровне сложности продажа привилегированных аций почти никогда не ведет к победе. Более того в Англии и в Европе для успеха фирмы совершенно необходимо как можно быстрее возвратить первоначально взятый долг.

7. Вы можете иметь доход от любого бизнеса - от спекуляций на бирже, от исполнения приоритетных экспресс-доставок, но очень скоро поймете, что если Ваша основная деятельность (грузоперевозки) выполняется неэффективно, то Вам ничто не поможет, и бизнесменом Вам не быть.

Уважаемые читатели!
К сожалению, в № 2 за 1992 год в статье "Опыт работы с DR DOS 5.0" на стр.5 нами допущена досадная ошибка. Фирма IBM НЕ ПОСТАВЛЯЕТ *компьютеры с операционной системой DR DOS. Фирма IBM поставляет свои компьютеры с операционной системой IBM DOS, а также с операционной системой OS/2. Редакция и автор приносят вам свои извинения и выражают благодарность сотруднику фирмы IBM Михаилу Васильеву, обратившему наше внимание на эту ошибку.*

Уважаемые господа!
Акционерное общество "ЮниВер" предлагает Вам:

ОС

Программное обеспечение для новичков и профессионалов.

- Более 300 лицензионных программных продуктов ведущих зарубежных и отечественных фирм - от игр и обучающих комплексов до профессиональных пакетов.

СУБД

Вычислительная техника и периферийное оборудование.

- CANON NAVIGATOR - компьютер, факс, ксерокс, сканер, струйный принтер, автоответчик - в едином комплексе;
- компьютеры IBM PC/AT 386;
- принтеры Microline;
- коммуникационные средства;
- расходные материалы.

САПР

ДГ

Разработка информационных систем "под ключ".

- Коллектив программистов высокого класса занимается разработкой программного обеспечения любой степени сложности.

Услуги пользователям:

- предпроектное обследование и разработка технических заданий;
- разработка и сопровождение программного обеспечения;
- установка программного обеспечения, генерация сетей персональных ЭВМ;
- обучение и консультации.

ЯП

ИГРЫ

Оплата в любой удобной для Вас форме!

Адреса наших магазинов в Москве:
пр-т Вернадского, 101, (ст. м. "Юго-Западная"),
ул. Новый Арбат, 26, Дом книги, 2-й этаж (ст. м. "Арбатская"),
ул. Мясницкая, 6, "Книжный мир" (ст. м. "Лубянка")

Телефоны: (095) 434-30-69, 434-46-20
Факс: (095) 434-46-20, 434-20-60

А. Поликарпов

EGA и VGA в текстовых режимах

Основой, "скелетом" интерактивных программ является пользовательский интерфейс. Однако разработка такого интерфейса, сама по себе являющаяся сложной задачей, зачастую упирается в программирование средств отображения информации. Кроме того, существует еще ряд приложений, при программировании которых не обойтись без хорошего знания видеосистем ПК, например: отображение картографической информации, обработка видеоизображений и пр. Видеоадаптеры ПК IBM PC и IBM PS/2, начиная примерно с 1985 г., когда фирмой IBM был представлен адаптер EGA, позволяют успешно решать эти проблемы. Однако у нас в стране с информацией по аппаратной части ПК вообще и их видеосистемам в частности, мягко говоря, не густо. Использование средств, предоставляемых стандартными библиотеками для Turbo Pascal, Turbo C и т.д. и ориентированных в основном на широкого пользователя, и уж тем более видеофункций BIOS, не всегда приносят желаемый результат. Поэтому предлагаемая статья, наверное, будет интересна всем программистам, работающим со средствами отображения информации, а также тем, кому просто интересно знать, как работают видеосистемы ПК IBM PC. Необходимо сразу оговориться, что здесь описаны только адаптеры EGA (Enhansed Graphics Adapter) и VGA (Video Graphics Array), как наиболее распространенные в нашей стране. Кроме того, рассматривается только текстовый режим их работы, так как графика - предмет для отдельного, очень большого разговора.

Функциональные компоненты видеосистем EGA и VGA

Основными функциональными компонентами видеоадаптеров EGA и VGA являются видеобуфер, ПЗУ и набор контроллеров: видеоконтроллер, задатчик последовательностей, контроллер атрибутов, графический контроллер и знакогенератор.

Видеопамять (видеобуфер)

Видеопамять - область ОЗУ, расположенная на плате видеоадаптера и предназначенная для хранения отображаемых на экране монитора данных. Она включена в адресное пространство микропроцессора и поэтому доступна для непосредственной записи/считывания. Адрес начала видеопамяти программируется и для текстового режима составляет B800:0000. Обьем памяти может быть различен. Как правило, на платах EGA и VGA устанавливается 256K памяти, что достаточно для хранения нескольких страниц экрана.

Контроллеры

Контроллеры - специальная логика, выполняющая те или иные функции по интерпретации данных и управлению видеосистемой. Все они имеют набор внутренних регистров, содержимое которых задает режим работы видеоконтроллера. Почти все внутренние регистры доступны для программирования через соответствующие порты.

Видеоконтроллер

Видеоконтроллер генерирует основные сигналы синхронизации (вертикальной и горизонтальной) и обеспечивает совместную работу всех подсистем адаптера. Он управляет счетчиком адреса, синхронизируя его увеличение со скоростью считывания, декодирования и вывода информации на экран. Кроме того, видеоконтроллер выполняет ряд других функций, в частности определяет адрес, начиная с которого информация из видеобуфера отображается на экране, контролирует размеры и положение курсора на экране и др.. Видеоконтроллер имеет адресный регистр, регистр статуса и 25 регистров данных. Адресный регистр доступен через порт 3B4h для монохромного и 3D4h для цветного режимов. Регистр статуса - через порт 3BAh для монохромного и 3DAh для цветного режима, а регистры данных - через порты 3B5h и 3D5h соответственно. Общий интерфейс программирования видеоконтроллера приведен в примере 1. Сначала Вы записываете в порт 3x4h номер требуемого регистра данных, а затем в порт 3x5h - сами данные. Названия регистров видеоконтроллера приведены в табл.1. Все регистры данных видеоконтроллера (кроме 0Ch, 0Dh, 0Eh и 0Fh) на адаптере EGA доступны только по чтению, на VGA - и по чтению, и по записи. Регистр статуса только читается.

Пример 1

```
MOV   DX,  3D4h
MOV   AL,  Номер_Требуемого_Регистра
OUT   DX,  AL
INC   DX
MOV   AL,  Данные
OUT   DX,  AL
```

Таблица 1. Регистры видеоконтроллера

Адрес порта	Номер регистра	Назначение регистра
3BAh (3DAh)		Статусный регистр
3B4h (3D4h)		Регистр адреса
3B5h (3D5h)		Регистры данных
	00h	По горизонтали всего
	01h	Размер отображаемого по горизонтали
	02h	Начало невидимой области по горизонтали
	03h	Конец невидимой области по горизонтали
	04h	Начало обратного хода луча
	05h	Конец обратного хода луча
	06h	По вертикали всего
	07h	Регистр переполнений
	08h	Начало видимой области по вертикали
	09h	Размер символа по вертикали
	0Ah	Начало курсора
	0Bh	Конец курсора
	0Ch	Начальный адрес видеобуфера(ст. байт)
	0Dh	Начальный адрес видеобуфера(мл. байт)
	0Eh	Положение курсора (ст. байт)
	0Fh	Положение курсора (мл. байт)
	10h	Начало обратного хода по вертикали
	11h	Конец обратного хода по вертикали
	12h	Конец видимой области по вертикали
	13h	Логическая длина линии экрана
	14h	Положение подчеркивания
	15h	Начало невидимой области по вертикали
	16h	Конец невидимой области по вертикали
	17h	Управление режимом
	18h	Сравнение строк

Задатчик последовательностей

Если видеоконтроллер обеспечивает, в основном, внешнюю синхронизацию, то задатчик последовательностей генерирует синхросигналы "внутреннего потребления", предназначенные для организации циклов записи/считывания видеопамяти, т.е. позволяет процессору обращение к ней только во время активных циклов вывода информации на экран. Тем самым видеоадаптер ликвидирует появление интерференции ("снега") на экране, что было характерно для CGA. Кроме того, он позволяет защищать отдельные банки видеопамяти от записи, а также выполняет ряд других функций. Задатчик последовательностей, как и видеоконтроллер, имеет адресный регистр, доступный через порт 3C4h, и 5 регистров данных, доступных через порт 3C5h. Программирование задатчика последовательностей подобно программированию видеоконтроллера. Описание регистров задатчиков последовательностей приведено в табл. 2.

Таблица 2. Регистры задатчика последовательностей

Адрес порта	Номер регистра	Назначение регистра
3C4h		Регистры адреса
3C5h		Регистры данных
	00h	Регистр сброса
	01h	Регистр режима синхронизации
	02h	Регистр маски записи в видеопамять
	03h	Регистр выбора знакогенератора
	04h	Регистр режима памяти

Контроллер атрибутов

Контроллер атрибутов выполняет функции, связанные с управлением цветами и интерпретацией атрибутов символа. Он имеет 21 внутренний регистр, которые доступны через порт 3C0h. Описание регистров приведено в табл.3.

Таблица 3. Регистры контроллера атрибутов

Адрес порта	Номер регистра	Назначение регистра
3C0H	00-0Fh	Регистры палитры
	10h	Управление режимом
	11h	Цвет окантовки
	12h	Регистр разрешения основного цвета
	13h	Регистр горизонтального сдвига изображения

Способ программирования его несколько сложнее, чем для других контроллеров, так как он не имеет отдельного адресного регистра. Вместо этого имеется внутренний триггер, который переводит порт из адресного режима в режим записи данных. Для того чтобы перевести его в адресный режим, необходимо выполнить операцию ввода/вывода с портом статусного регистра видеоконтроллера. Пример 2 иллюстрирует программирование контроллера атрибутов.

Пример 2

```
MOV   DX,  03DA              ; перевод в адресный режим
IN    AL,  DX
MOV   DX,  03C0h             ; порт контроллер а атрибутов
MOV   AL   номер_регистра    ; адресация
OUT   DX,  AL
MOV   AL,  данные            ; запись данных
OUT   DX,  AL
```

Но и это не все. Переводя контроллер атрибутов в адресный режим, Вы тем самым запрещаете ему доступ к внутренним регистрам (так как бит 5 при записи адреса в порт 3C0h сброшен в 0). Чтобы разрешить контроллеру использовать те данные, которые Вы записали, необходимо опять перевести контроллер в адресный режим и разрешить ему доступ к данным, установив бит 5 в 1 (пример 3).

Пример 3

```
MOV   DX,  03DAh             ; перевод в адресный режим
IN    AL,  DX
MOV   DX,  03C0h             ; порт контроллера атрибутов
MOV   AL,  020h              ; разрешить доступ
OUT   DX,  AL
```

ПЗУ

ПЗУ - область постоянной памяти, расположенная на плате адаптера и включенная в адресное пространство процессора, начиная с адреса C000:0000. ПЗУ содержит ряд таблиц, необходимых для работы адаптера, а также видеоBIOS - набор функций, обеспечивающих единый программный интерфейс для работы с видеосистемой. При включении ПК система инициализируется таким образом, что ряд векторов, в частности вектор 10h, указывает именно на видеоBIOS.

ВидеоBIOS - довольно мощный инструмент в умелых руках. Некоторые стандартные функции, в частности установка режима, загрузка регистров палитры и др., обычно тяжелы для программирования и редко выполняются, и поэтому есть прямой смысл не изобретать велосипед. В общем правило такое: там, где это не снижает эффективности программы, лучше использовать средства BIOS.

Основы работы с видеоадаптерами EGA и VGA в текстовом режиме

Текстовый режим работы видеосистем наиболее распространен в существующем программном обеспечении. К числу его несомненных преимуществ относится высокая скорость вывода инфор-

мации, так как практически всю работу за Вас выполняет аппаратная часть. Кроме того, программы, работающие в текстовом режиме, имеют дело с гораздо меньшим объемом данных, чем в графике. Наконец, значительно увеличивается совместимость программ, так как не все адаптеры поддерживают, например, графический режим с разрешением 640x480 точек, но все имеют текстовый режим 80x25 символов.

Для того чтобы писать хорошие, эффективные программы вывода информации на экран в текстовом режиме на видеоадаптерах EGA и VGA, необходимо ответить на четыре следующих вопроса.

1. Как перевести видеоадаптер в требуемый режим работы ?

2. Как организован видеобуфер, т.е. в каком формате хранятся видеоданные и как адресоваться к этим данным ?

3. Как управлять цветом ?

4. Как управлять курсором ?

Установка требуемого режима

Вообще говоря, для того чтобы настроить видеоадаптер на определенный режим работы, необходимо выполнить довольно большую работу по программированию всех контроллеров. Однако есть гораздо менее трудоемкий и более безопасный путь - воспользоваться функцией 0 (установить режим) видеоBIOS.

ВидеоBIOS поддерживает 5 стандартных текстовых режимов (табл.4). Режимы 0 и 2 предназначены для работы с композитными мониторами, которые ни EGA, ни VGA не поддерживают. Поэтому для этих адаптеров режимы 0 и 1, 2 и 3 равнозначны. Режим 7 предназначен для работы с монохромным монитором.

Таблица 4. Стандартные текстовые режимы видеосистем EGA и VGA

N	Разрешение, символ	Разрешение, пиксел	Размер знакоместа	Количество цветов	Адаптеры	Адрес начала буфера
0	40x25	320x200	8x8	16/8	EGA,VGA	b800
0	40x25	320x350	8x14	16/8	EGA,VGA	b800
0	40x25	360x400	8x16	16/8	VGA	b800
1	40x25	320x200	8x8	16/8	EGA,VGA	b800
1	40x25	320x350	8x14	16/8	EGA,VGA	b800
1	40x25	360x400	8x16	16/8	VGA	b800
2	80x25	640x200	8x8	16/8	EGA,VGA	b800
2	80x25	640x350	8x14	16/8	EGA,VGA	b800
2	80x25	720x400	8x16	16/8	VGA	b800
3	80x25	640x200	8x8	16/8	EGA,VGA	b800
3	80x25	640x350	8x14	16/8	EGA,VGA	b800
3	80x25	720x400	8x16	16/8	VGA	b800
7	80x25	720X350	9x14	3	EGA,VGA	b000
7	80x25	720X40	9x16	3	VGA	b000

Для того чтобы перевести адаптер в требуемый видеорежим, необходимо выполнить прерывание 10h (стандартный способ вызова функций видеоBIOS), предварительно записав в регистр AL номер требуемого режима и обнулив регистр AH (номер функции). При этом BIOS соответствующим образом запрограммирует контроллеры, очистит видеобуфер и установит текущей 0-ю страницу памяти. Однако Вы можете контролировать набор операций, которые BIOS выполняет, изменив некоторые ячейки в области данных видеоBIOS, расположенные в ОЗУ с адреса 0040:0049 или используя функцию 12h (конфигурацию видеосистемы) через прерывание 10h. Кроме того, устанавливая бит 7 в AL в 1 перед вызовом функции 0, Вы отменяете операцию очистки видеопамяти при установке режима (пример 4).

Пример 4

```
XOR   AH,  AH        ; номер функции - 0
MOV   AL,  03h       ; номер режима - 3
OR    AL,  80h       ; не очищать видеобуфер
INT   10h            ; вызов функции видеоBIOS
```

Организация видеобуфера в текстовом режиме

Хотя физическая организация видеобуфера в адаптерах EGA и VGA достаточно сложна, в большинстве случаев программист может представить ее себе (по аналогии с обычным ОЗУ) в виде линейной последовательности байтов, начинающейся по определенному адресу и имеющей определенную длину. Все, что содержится в этой области, может быть отображено на мониторе.

Каждому символу на экране соответствует его "образ" в видеопамяти, состоящий из двух байтов: байта атрибута и байта ASCII кода символа, причем байт кода расположен по четному адресу, а байт атрибутов - по следующему, нечетному, адресу. Видеоадаптер с частотой 50 - 70 Гц считывает данные из видеобуфера и посылает байт атрибутов в контроллер атрибутов, который вырабатывает соответствующие цветовые сигналы, а байт кода - в знакогенератор, который выбирает соответствующую коду символа матрицу с изображением символа, записанную в таблице в видеобуфере, и посылает ее в формирователь видеосигналов. В нем, в соответствии с полученнными от знакогенератора данными и цветовыми сигналами от контроллера атрибутов, вырабатывается последовательность сигналов, управляющих монитором. Видеоконтроллер и задатчик последовательностей обеспечивают правильную адресацию при считывании данных из видеобуфера и формировании видеосигналов.

Зная, что каждый символ занимает в видеопамяти 2 байта, нетрудно подсчитать, что для хранения 1 страницы экрана необходимо 80x25x2 = 4000 байтов в режиме 80x25 символов и 40x25x2 = 2000 байтов в режиме 40 x25 символов. Однако будет ошибкой думать, что если Вы имеете на плате 256K, то это позволит Вам организовать 256*1024/4000 = 65 страниц. Контроллеры программируются таким образом, что в текстовом режиме одновременно доступны под "образы" символов только 32K видеопамяти, начиная с B000:0000 для режима 7 и с B800:0000 для режимов 0 - 3. Остальная память в обычном режиме недоступна программисту. Таким образом, в режиме 80x25 символов можно организовать только 32*1024/4000 = 8 страниц длиной в 4000 байтов.

Вы можете изменить начальный адрес отображаемой области, записав в регистры 0Ch и 0Dh видеоконтроллера смещение до того "образа", начиная с которого вы хотите отображать данные. Одновременно необходимо то же значение записать в ячейку 0040:004 в области данных видеоBIOS, чтобы не конфликтовать с системой.

Зная, как организованы данные в видеобуфере, нетрудно понять, как адресоваться к конкретному символу на экране, расположенному в строке R и колонке C экрана. В общем случае формула для расчета адреса "образа" этого символа выглядит следующим образом:

Offset = Start + (R * B + C) * 2,

где B - количество символов в строке;

Start - смещение относительно начала видеобуфера до первого символа отображаемой области (то, что записано в регистрах 0Ch и 0Dh видеоконтроллера или в ячейке 0040:004);

Offset - требуемое смещение относительно начала видеобуфера.

Здесь за начало координат берется верхний левый угол с координатами (0,0).

Управление цветом

Управление цветом, пожалуй, наиболее необходимая на практике область работы с видеоадаптером. И EGA, и VGA обладают хорошими характеристиками в этом плане. Так, например, EGA может одновременно отображать до 16 цветов из 64 возможных, VGA - уже до 256 из набора в 256 К. Вы можете за несколько операций изменить либо всю палитру, либо только ее часть, заставить VGA медленно тушить экран и еще делать много других фокусов. Все это, оказывается, не так уж сложно.

Интерпретация байта атрибутов

Байт атрибутов представляет собой 2 группы по 4 бита (рис. 1). Младшая группа, включающая биты 0 - 3 (младший полубайт), определяет цвет символа, причем биты 0 - 2 управляют основными цветами : красным (R), зеленым (G) и синим (B). Если соответствующий бит установлен в 1, то эта составляющая присутствует в цвете символа, а если сброшен в 0 - отсутствует.

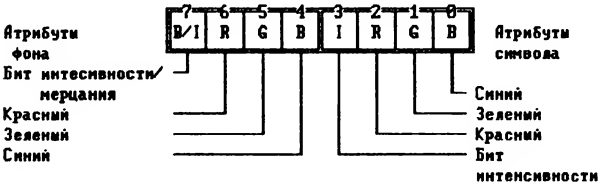


Рис. 1. Структура байта атрибутов

Таким образом, управляя включением/выключением этих составляющих, можно получить гамму из 8 цветов. Бит 3 в младшем полубайте служит для управления интенсивностью. Установка этого бита в 1 повышает яркость основных цветов, что дает возможность получить дополнительно 8 цветов.

Старший полубайт, определяющий цвет фона, интерпретируется аналогично, то есть биты 4 - 6 управляют основными составляющими в цвете фона. 7-й бит интерпретируется по-разному, в зависимости от того, как запрограммирован видеоадаптер. В одном случае он интерпретируется как бит интенсивности и тогда можно получить возможность отображать до 16 цветов в фоне. В другом случае этот бит управляет режимом мерцания, и тогда для фона становятся доступны только 8 цветов, но появляется возможность заставить символ мигать, причем, так как при этом задействуется только аппаратная часть адаптера, на управление миганием не расходуется ни такта процессорного времени.

Произведя небольшие подсчеты в уме, Вы наверное удивитесь: получается, что можно отобразить только 16 цветов. Откуда же число 64 для EGA и уж тем более 256 для VGA ?

EGA

Для получения 64 цветов с помощью двоичных сигналов необходимо иметь 6 линий управления. Действительно EGA - совместимые мониторы управляются 6 сигналами цвета: 3 основными (R, G, B) и 3 вторичными пониженной интенсивности (r, g, b). За генерацию соответствующих сигналов в адаптере отвечает контроллер атрибутов. Формат отдельных регистров приведен на рис. 2.

Познакомившись в общих чертах с управляющими элементами контроллера атрибутов, перейдем к тому, как обрабатывается этим контроллером байт атрибутов.

При этом происходит следующее. Четыре бита, определяющие цвет символа или фона, сначала при помощи операции AND маскируются содержимым четырех младших битов регистра разрешения цветовой плоскости. Полученное 4-битовое значение используется для выбора регистра палитры. И уже 6 -битовое число, записанное в этом регистре, определяет сигналы основных 6 цветовых составляющих для управления монитором. Вот и получается, что можно выбрать и отобразить одновременно до 16 цветов из 64 возможных.

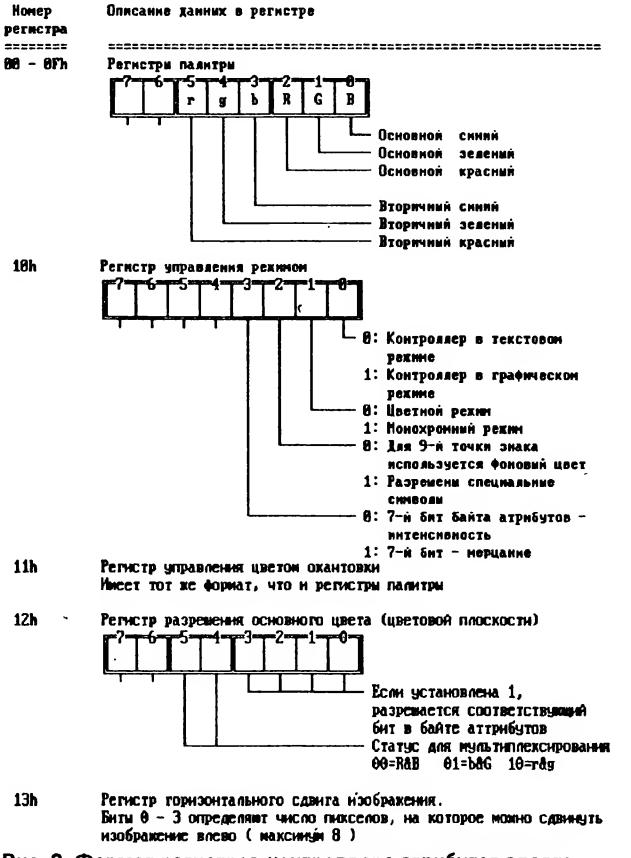


Рис. 2. Формат регистров контроллера атрибутов адаптеров EGA и VGA

VGA

В схему управления цветом VGA по сравнению с EGA включен новый элемент - видеоЦАП, что и позволяет значительно улучшить его возможности по отображаемой цветовой гамме.

Контроллер атрибутов с точки зрения программиста такой же, как и у EGA: он доступен через тот же порт, аналогично программируется, но есть и некоторые отличия. Во-первых, добавился новый регистр - регистр выбора цвета (его номер - 14h). Во-вторых, у VGA вы можете не только писать в регистры, но и читать из них. Наконец, у VGA является информационным бит 7 регистра управления режимом.

ВидеоЦАП имеет 256 цветовых 18-разрядных регистров, в которых записаны 6-разрядные числа для красной, синей и зеленой составляющих (рис.3). ВидеоЦАП преобразует 6-разрядное число для каждой составляющей в аналоговый сигнал. Кроме цветовых регистров видеоЦАП имеет 16 регистров палитры размерностью 6 битов каждый.

Процесс преобразования атрибута в управляющие сигналы похож на такой же процесс у EGA. 4-битовый атрибут, определяющий цвет символа или фона, маскируется четырьмя битами регистра разрешения цветовой плоскости. Полученное в ре-

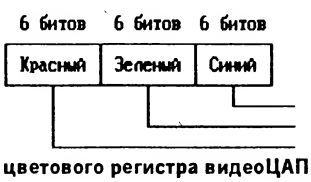


Рис. 3. Формат цветового регистра видеоЦАП VGA

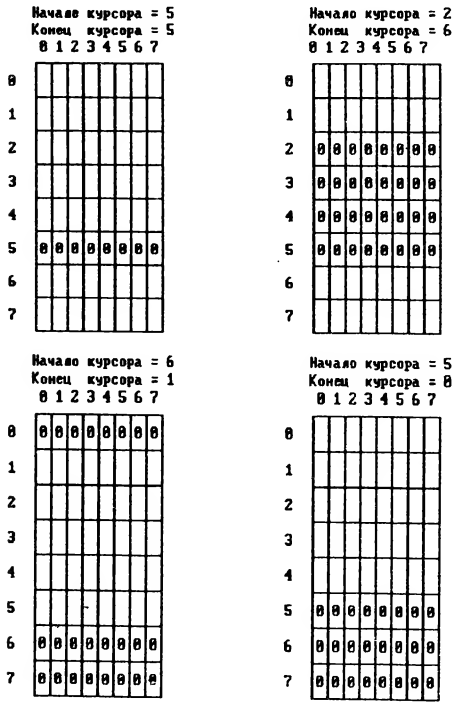


Рис. 4. Курсор EGA монитора

зультате 4-битовое число используется для выбора одного из 16 регистров палитры. Далее все зависит от значения бита 7 регистра управления режимом контроллера атрибутов.

Если бит 7 сброшен в 0, то контроллер атрибутов комбинирует 6- битовый регистр палитры с битами 2 и 3 регистра выбора цвета. Получаемое 8-битовое число, маскируясь содержимым масочного регистра видеоЦАП, используется для выбора одного из 256 цветовых регистров видеоЦАП, который уже и определяет комбинацию управляющих сигналов. Таким образом, в этом случае все 256 цветовых регистра видеоЦАП разбиваются на 4 группы по 64 регистра. 6-битовый регистр палитры используется для выбора 1 из регистров в группе, а биты 2 и 3 регистра выбора цвета определяют, какая группа используется.

Если бит 7 установлен в 1, то используются только 4 младших бита в регистре палитры. Контроллер атрибутов берет остальные 4 бита из бит 0-3 регистра выбора цвета. Далее все происходит аналогично сброшенному биту 7 . Здесь уже цветовые регистры видеоЦАП разбиваются на 16 групп по 16 регистров, и как и ранее регистр палитры выбирает 1 из цветов в группе, а биты 0-3 регистра выбора цвета определяют одну из групп. Этот режим декодирования атрибута делает возможным один из наиболее известных эффектов адаптера VGA: медленное "угасание" экрана. Для этого необходимо запрограммировать цветовые регистры 16 цветами стандартной палитры, только каждая следующая группа должна отличаться от предыдущей более низкой интенсивностью цвета. После этого, переключая палитры простым изменением содержимого регистра выбора цвета контроллера атрибутов и вводя необходимую задержку между этими переключениями, Вы можете получить указанный эффект.

Управление курсором

Курсором на адаптерах EGA и VGA "ведает" видеоконтроллер, который имеет 4 специальных регистра: регистры положения курсора (старший и младший байт) - 0Eh и 0Fh соответственно, регистр начала и конца курсора - 0Ah и 0Bh соответственно.

В регистры 0Eh и 0Fh записывается смещение относительно начала видеобуфера до того "образа" символа, на месте которого Вы хотите разместить курсор. Поэтому, если Вы меняете начальный адрес отображаемой области, изменив содержимое регистров 0Ch и 0Dh, Вам необходимо изменить и содержимое 0Eh и 0Fh. Отсюда следует один из способов сделать курсор невидимым : достаточно просто "загнать" его в невидимую область. Как правило, его загоняют на 25 строку. Размеры и форма курсора определяются содержимым регистров 0Ah и 0Bh. При этом в регистр начало курсора (0Ah) записывается номер строки матрицы символа, с которой начинается курсор. Интерпретация же регистра конец курсора (0Bh) на EGA и VGA несколько различна.

EGA

На EGA в регистр 0Bh Вы должны записать число, на 1 большее, чем номер последней строки курсора (см. рис.2). Если содержимое регистра 0Ah меньше, чем регистра 0Bh, то курсор "заворачивается" вокруг матрицы. Если они равны, то курсор занимает только строку, определенную регистром 0Ah. И последнее: значение регистра 0Bh должно быть меньше числа линий в строке, иначе курсор "размажется" по всей матрице. Курсор EGA монитора дан на рис. 4.

VGA

На VGA в регистре 0Bh должен быть записан именно номер последней линии курсора в матрице. Кроме того, если содержимое 0Bh меньше содержимого 0Ah, он не "проворачивается", а исчезает. Курсор VGA приведен на рис. 5.

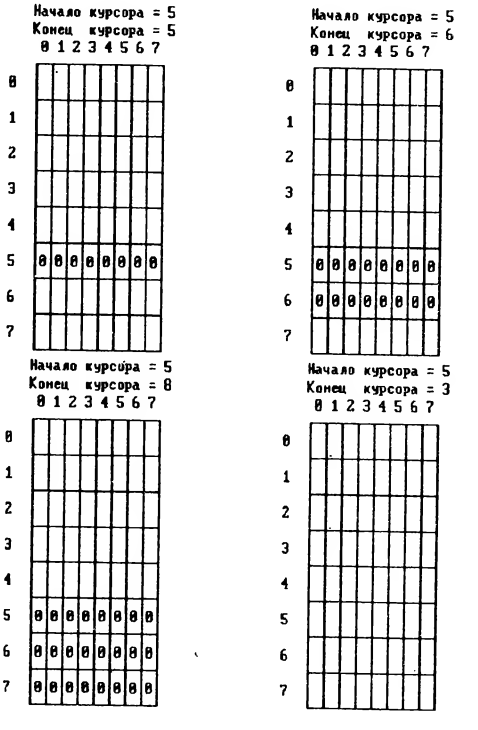


Рис. 5. Курсор VGA монитора

Паскаль + Ассемблер = ... или как использовать "мышь" в графических режимах

В.Водолазкий, В.Головин, Е.Короткова

Turbo Assembler предоставляет широкие и мощные средства, позволяющие добавлять коды на языке ассемблера в программу на Паскале. В этой статье указанная возможность демонстрируется на примере реализации программного интерфейса позиционного манипулятора "мышь" из программ на Turbo Pascal. Хотим отметить, что в настоящее время распространенные библиотеки процедур Turbo Professional и Turbo Vision (версия 6.0) поддерживают работу с "мышью" только в текстовых режимах. В то же время графические возможности Turbo Pascal по-прежнему ограничиваются разработанным несколько лет назад фирмой Borland набором процедур и функций, получившим название Borland Graphics Interface (BGI) и не претерпевшим с версии 4.0 никаких изменений. Имеющиеся коммерческие пакеты из-за высокой стоимости практически недоступны широкому кругу пользователей. Поэтому каждый вынужден составлять собственную библиотеку для решения своих задач.

Большинство программ, которые вам могут понадобиться, можно написать просто на Паскале. В отличие от большинства версий Паскаля Turbo Pascal допускает произвольный доступ ко всем машинным ресурсам через массивы Port[], Mem[], MemW[] и MemL[], а также разрешает обращения к BIOS и операционной системе при помощи процедур Intr() и MsDos().

Зачем же тогда обращаться из Turbo Pascal к Turbo Assembler? Для этого имеются три наиболее вероятных причины: выполнение некоторых немногочисленных операций, недоступных непосредственно из Turbo Pascal; необходимость в максимально высоком быстродействии программы, достижимая только в языке ассемблера; минимизация объема кода.

Ну хорошо, скажет дотошный читатель, а как же обеспечить стыковку программ на ассемблере с процедурами на Паскале? Здесь фирма Borland пошла по пути реализации идеи "консенсуса" : определенные шаги сделаны как со стороны Turbo Assembler по направлению к Turbo Pascal, так и наоборот. Рассмотрим эти вопросы подробнее.

Слушай секретное слово: .MODEL TPASCAL

Вначале рассмотрим возможности, предоставляемые ассемблером. Директива .MODEL с параметром TPASCAL устанавливает упрощенную сегментацию, модель памяти и поддержку языка. Использование TPASCAL с директивой .MODEL устанавливает соглашения о связях Паскаля, определяет имена сегментов, выполняет команды PUSH BP и MOV BP,SP, а также устанавливает возврат при помощи команд POP BP и RET N (где N - длина параметров в байтах). Директива Proc позволяет определить ваши параметры в том же порядке, в котором они определены в программе на Паскале. Это позволяет значительно упростить написание функций на ассемблере и сократить время на отладку за счет автоматического построения входных и выходных фрагментов в каждой из процедур. Использование модели TPASCAL позволяет вам эффективно использовать возможности процессора 80286 путем установки директивы .286c в начале программы на ассемблере. В этом случае компилятор автоматически подключает операторы ENTER и LEAVE.

Передача данных в ассемблерные процедуры осуществляется по следующим правилам.

Если данные должны возвращаться в вызывающую процедуру (объявления VAR в описании процедуры или функции), в ассемблерную процедуру передаются дальние указатели на соответствующую структуру данных (DWORD).

Если аргументы передаются только в одном направлении, используется следующая таблица соответствия (табл.1).

Таблица 1.Соответствие типов аргументов

Объявления типов аргументов	
В программе на Паскале	В программе на ассемблере
char	byte
byte	byte
integer	word
word	word
real	dword
single	dword

Приведенные выше данные передаются по значению. Все остальные типы данных (строки, множества, вещественные ...) передаются по ссылке, т.е. в процедуру передается дальний указатель на соответствующую переменную.

Подключение объектных файлов к программам на Паскале

Процедуры и функции, написанные на ассемблере, могут быть связаны с программами и модулями Turbo Pascal с помощью директивы компилятора $L. Исходный файл на ассемблере транслируется в объектный файл (.OBJ) с помощью TurboAssembler. При этом, используя несколько директив $L, вы можете с программой или модулем связать несколько объектных файлов.

Процедуры и функции , написанные на ассемблере, должны быть объявлены в программе или модуле на Паскале как external, например:

```
Function CheckMouse : boolean; external;
```

При этом необходимо помнить о важном ограничении Паскаля: не допускается использование в программе на языке ассемблера констант в секции .DATA. Если же вам все-таки необходимо объявить в процедуре на ассемблере инициализированные переменные или блок констант, разместите их в кодовом сегменте с обязательной настройкой регистров DS или ES.

Разрабатываем курсор для "мыши"

Описание процедур и функций, включенных в состав библиотеки, приведено ниже. Мы хотим обратить внимание только на один момент: разработку собственного графического курсора для "мыши". При инициализации или программном сбросе драйвера "мыши" происходит автоматическая установка курсора, определенного в теле драйвера. В то же время вам может понадобиться использовать свой курсор специального вида, который, например, может информировать пользователя об определенном состоянии программы. Например, когда среда Microsoft Windows занята, курсор на экране приобретает форму песочных часов. Но ведь и мы не хуже. Для определения собственного курсора "мыши" необходимо сделать не так уж много : заполнить два массива шаблонов 16 по 16 точек (курсора и экрана) и определить горячую точку курсора. Пример описания курсоров приведен в тексте библиотеки процедур. Отметим, что цвет точки на экране при отображении курсора определяется по следующим правилам (табл.2).

Таблица 2.Правила определения цвета точки на экране

Цвет точки	Черный	Фоновый	Инверсный	Белый
Шаблон экрана	0	1	1	0
Шаблон курсора	0	0	1	1

Горячая точка курсора определяет, какое смещение будет добавлено к координатам левого верхнего угла маски курсора при считывании его координат. Зачем это нужно? Для удобства в работе. С помощью настройки горячей точки курсора вы можете потребовать, чтобы при использовании стреловидного курсора считывались координаты конца стрелки, при использовании курсора-прицела—координаты точки в перекрестии и т.д. Это избавит вас от лишней работы при обработке данных от "мыши".

Библиотека для работы с "мышью"

Помня об ограничении области глобальных данных в 64 Кбайта и не желая вносить пользователю какие-либо ограничения, весь интерфейс библиотечных функций реализован через локальные переменные, динамически размещаемые в операционном стеке при вызове соответствующей функции.

Процедуры и функции для обслуживания "мыши"

1) Function CheckMouse : boolean;
   Проверяет наличие в системе драйвера "мыши". Возвращает true, если драйвер "мыши" в системе установлен, и false в противном случае.
2) Function InitMouse : boolean;
   Осуществляет инициализацию аппаратных средств и драйвера "мыши". Должна вызываться перед началом работы с устройством. Возвращает true, если инициализация прошла успешно, и false, если отсутствуют аппаратные или программные средства.
3) Procedure SetMouseCursorOn;
   Сделать курсор "мыши" видимым.
   Если курсор невидим, то он включается.
4) Procedure SetMouseCursorOff;
   Сделать курсор "мыши" невидимым. Эту функцию необходимо вызывать каждый раз перед тем, как произвести какие-либо изменения на том участке экрана, где находится курсор. Кроме того, рекомендуется вызывать эту процедуру в конце работы программы.
5) Procedure ReadMouseStatus(VAR hor,vert : word;
   VAR b1,b2,b3 : boolean);
   Прочитать состояние "мыши".

Возвращаемые значения: hor,vert — горизонтальная и вертикальная координаты курсора;
b1 — левая кнопка (true - нажата);
b2 — правая;
b3 — средняя.

6) Procedure SetMouseCursor(hor,vert : word);
Установить курсор на экране в заданную позицию:
hor — горизонтальная координата;
vert — вертикальная координата.

7) Procedure MouseRangeX (leftg,rightg : word);
Установить диапазон перемещения курсора по горизонтали (X):
leftg, rightg - минимальная и максимальная границы координат курсора.

8) Procedure MouseRangeY (leftg,rightg : word);
Установить диапазон перемещения курсора по вертикали (Y):
leftg,rightg - минимальная и максимальная границы координат курсора.

9) Procedure SetGraphMouseShape(shape : word);
Установить параметры графического "мышиного" курсора. shape - выбор типа курсора из встроенной библиотеки. Система автоматически настраивает координаты горячего пятна. Размеры курсора 16 по 16 точек. В случае некоректного задания номера курсора устанавливается курсор номер 1. В настоящей версии реализованы 2 варианта графического курсора.

10) Procedure SaveMouseStatus;
Сохранить текущее состояние "мыши" во внутреннем буфере. При повторном вызове предыдущее состояние стирается.

11) Procedure RestoreMouseStatus;
Восстановить предварительно сохраненное состояние драйвера "мыши".

12) Procedure SetMouseIntFreq(b: byte);
Установить частоту прерываний "мыши".
Значения параметра:
0 - прерывания запрещены;
1 - 30 прерываний в секунду;
2 - 50 прерываний в секунду;
3 - 100 прерываний в секунду;
4 - 200 прерываний в секунду.
В случае установки других значений осуществляется их преобразование к виду 4.

13) Function MouseReset : boolean;
Осуществляет программный сброс драйвера "мыши". По сути то же , что и InitMouse, но не осуществляет сброса аппаратуры "мыши". Возвращает true, если установка успешна, и false, если драйвер "мыши" в памяти не установлен.

14) Procedure ReadMouseCounters(var x,y : integer);
Осуществляет считывание числа сигналов "микки", накопленное внутренними счетчиками драйвера "мыши" с момента последнего вызова этой функции. Положительные значения соответствуют движению вправо или вниз. После вызова счетчик обнуляется. Диапазон значений - от -32768 до 32767.
Возвращает:
X - смещение по горизонтали;
Y - смещение по вертикали.
Обратите внимание, что эта процедура возвращает значения типа INTEGER, а не WORD, как большинство функций библиотеки.

15) Function LeftButtonPressed : boolean;
Проверяет состояние левой кнопки "мыши". Возвращает true, если в момент вызова функции кнопка нажата.

16) Function RightButtonPressed : boolean;
Проверяет состояние правой кнопки "мыши". Возвращает true, если в момент вызова функции нажата правая кнопка "мыши".

17) Function LeftButtonReleased : boolean;
Проверяет состояние левой кнопки "мыши". Возвращает true, если в момент вызова функции кнопка отпущена.

Programmer

Использование процедур на ассемблере обеспечивает:
—выполнение операций, трудно реализуемых средствами Pascal;
—достижение максимально возможного быстродействия;
—минимизацию объёма исполняемого кода

18) Function RightButtonReleased : boolean;
Проверяет состояние правой кнопки "мыши". Возвращает true, если в момент вызова функции кнопка отпущена.

Все процедуры и функции, перечисленные выше, приведенгы в листинге 1 (ассемблерная часть) и листинге 2 (интерфейсный модуль на Паскале). Листинг 3 содержит пример работы с "мышью" в графическом режиме. Краткая сводка функций управления "мышью" Microsoft mouse приведена в табл. 3.

Журнал "Монитор" приглашает к сотрудничеству:

- Дилеров по распространению журнала
- Специалистов по анализу состояния компьютерного рынка
- Авторов оригинальных статей и книг по программированию и вычислительной технике

Тел. (095)247-36-25.
Факс (095)126-31-56.
Адрес для писем: 117419, Москва, а/я 765.

Таблица 3. Краткая сводка функций управления "мышью" Microsoft mouse

Краткое описание функции	Передаваемые при вызове параметры	Возвращаемые значения
При использовании "мыши" с Геркулесом вы должны предварительно установить ячейку 0040:0049 в 6 для 0-й страницы и в 5 для страницы 1	AX = 0000	AX = 0000 "мышь" или драйвер отсутствуют BX = число кнопок FFFF - две кнопки 0000 - не две 0003 - "мышь" Mouse Systems/Logitech
Включить отображение курсора "мыши" на экране	AX = 0001	
"Спрятать" курсор "мыши". Имейте в виду, сколько раз вы вызовете эту процедуру, столько же должны вызвать предыдущую чтобы курсор вновь появился	AX = 0002	
Определяет текущее состояние "мыши". BX: бит 0 = 1 - нажата левая кнопка, бит 1 = 1 - нажата правая бит 2 = 1 - нажата средняя	AX = 0003	BX - состояние кнопок CX - координата X DX - координата Y
Установка курсора "мыши" в точку с координатами CX,DX	AX = 0004 CX = координата X DX = координата Y	
Функция возвращает данные о выбранной кнопке. AX : бит 0 - левая, бит 1 - правая, бит 2 - средняя	AX = 0005 BX = номер кнопки	AX = состояние BX = сколько раз была нажата кнопка CX = координата X последнего нажатия DX = координата Y
Функция возвращает информацию по числу отжатий выбранной кнопки с момента своего предыдущего вызова и координаты последнего события	AX = 0006 BX = номер кнопки	AX = состояние кнопки (см.выше) BX = число отпусканий данной кнопки CX = координата X DX = координата Y
Определение диапазона изменения координат курсора "мыши" по X (от Xmin до Xmax)	AX = 0007 CX = Xmin DX = Xmax	
Определение диапазона изменения координат курсора "мыши" по вертикали (от Ymin до Ymax)	AX = 0008 CX = Ymin DX = Ymax	
Определение графического курсора	AX = 0009 BX = смещение горячего пятна (X) CX = смещение горячего пятна (Y) ES:DX - битовый образ курсора	
Определение курсора для текстового режима. Текущие атрибуты экрана маскируются с маской экрана по AND, а затем по XOR с маской курсора	AX = 000A BX = 0000 - программный курсор; CX = маска экрана; DX = маска курсора	
Определение аппаратного курсора "мыши" для текстовых режимов отображения	AX = 000A BX = 0001 - аппаратный курсор CX = начальная строка сканирования DX = конечная строка сканирования	
Считывание накопленного сдвига с момента предыдущего вызова функции. Возвращаемые значения находятся в диапазоне от -32767 до 32767	AX = 000B	CX = накопленный сдвиг по горизонтали DX = накопленный сдвиг по вертикали
Функция задает внешнюю процедуру обработки прерывания от "мыши"	AX = 000C CX = маска вызова ES:DX - адрес процедуры обработки	
Включение эмуляции светового пера	AX = 000D	
Отключение эмуляции светового пера	AX = 000E	
Определение коэффициентов преобразования "микки" (шаг мыши) в пикселы на экране. По умолчанию установлены значения 8 - X, 16 - Y	AX = 000F CX = число "микки" на 8 пикселов по оси X DX = число "микки" на 8 пикселов по оси Y	
Определение зоны экрана для обновления. После вызова этой функции курсор выключается и должен быть активизирован явным вызовом функции 1	AX = 0010 CX,DX - координаты левого верхнего угла SI,DI - координаты правого нижнего угла	
Установка большого графического курсора. Вы можете задавать практически любой размер но (!) BH,HC - должны быть чётными, а -16<=BL,CL<=16	AX = 0012 BH,CH - ширина/высота курсора в словах BL,CL - координаты горячей точки ES:DX - адрес масок экрана и курсора	AX = FFFF, если установка успешна

Продолжение табл. 3

Краткое описание функции	Передаваемые при вызове параметры	Возвращаемые значения
Если скорость перемещения "мыши" по экрану превышает порог, то соотношение "микки"/пиксел увеличивается вдвое	AX = 0013 DX = порог срабатывания триггера	
Замена пользовательского обработчика прерываний "мыши", установленного ранее с помощью функции 000C, на новый	AX = 0014 CX = маска вызова (аналогична 000C) ES:DX - адрес обработчика	CX - маска предыдущего обработчика ES:DX - его адрес
Определение размера буфера,необходимого для сохранения данных драйвера "мыши"	AX = 0015	BX = размер буфера
Сохранение данных драйвера в буфере. Внимание! BX - не документирован, но должен обязательно устанавливаться	AX = 0016 BX = размер буфера ES:DX - адрес буфера	
Восстановление данных из буфера. При этом предварительно должен быть восстановлен видеорежим, соответствующий моменту сохранения данных в буфере	AX = 0017 BX = размер буфера ES:DX - адрес буфера данных	
Установка альтернативного обработчика прерываний от "мыши"	AX = 0018 CX- маска вызова ES:DX - адрес обработчика	AX = 0018 - установка успешная AX = FFFF - установка не удалась
Попытка найти обработчик пользователя, соответствующий маске вызова, заданной в CX	AX = 0019 CX = маска вызова	BX:DX - адрес обработчика CX - маска вызова (0, если не найдена)
Установка чувствительности "мыши"	AX = 001A BX - горизонтальная скорость CX - вертикальная скорость DX - порог удвоения скорости	
Считывание данных о чувствительности драйвера мыши	AX = 001B	BX = горизонтальная скорость CX = вертикальная скорость DX = порог удвоения скорости
Установка частоты опроса "мыши". Может использоваться только для манипуляторов "мышь", подключаемых к портам COM	AX = 001C BX = частота опроса: 0 - запрет прерываний 1 - 30 Гц, 2 - 50 Гц, 3 - 100 Гц, 4 - 200 Гц.	
Заказ определенного номера страницы дисплея. Курсор будет отображаться только на заданной странице	AX = 001D BX - номер страницы дисплея	
Считывание номера страницы дисплея, установленного предыдущей функцией	AX = 001E	BX - номер страницы дисплея
Отключение драйвера "мыши" с восстановлением векторов 10h и 17h (8086) или 74h (286/386)	AX = 001F	AX = 001F - операция выполнена AX = FFFF - не удалось ES:BX - настройка int 33 до установки
Восстановление драйвера мыши после его временного отключения предыдущей функцией	AX = 0020	
Программный сброс. Аналогичен функции 0000, но не инициализирует аппаратуру мышки и ее контроллера	AX = 0021	AX = FFFF - драйвер установлен AX = 0021 - драйвер отсутствует BX = 2, если драйвер установлен
Установка кода языка для сообщений	AX = 0022 BX - код языка: 0 - English, 1 - French, 2 - Dutch 3 - German, 4 - Swedish,5 - Finnish, 6 - Spanish, 7 - Portugese, 8 - Italian	
Возвращение кода языка, на котором выдаются сообщения (см. предыдущую функцию)	AX = 0023	BX - код языка

С целью предоставления владельцам персональных компьютеров информации о фирмах, действующих на компьютерном рынке, и предлагаемых ими продуктах и услугах, "Монитор" публикует на страницах журнала каталоги фирм и программ.

"Монитор" публикует также объявления фирм, желающих найти высококвалифицированных специалистов по компьютерам и программированию, и объявления частных лиц, ищущих работу в этой области.

Для получения более подробной информации Вы можете позвонить по телефону (095)247-36-25 либо выслать открытку со своими координатами по адресу 117419, Москва, а/я 765.

Листинг 1. Процедуры поддержки манипулятора "мышь", адаптированные для вызова из программ на Паскале

```
NAME MOUSE
PAGE    58,120          ;настраиваем длину страницы листинга
                        ;и ее ширину
.286C                   ;если у Вас PC/XT - удалите эту строку
.RADIX  16              ;константы по умолчанию будут вводиться
                        ;в 16-тиричном формате
;
;!-------------------------------------!
;! Библиотека процедур для работы с мышью PS/2 !
;! из программ на Turbo Pascal 4.0+ в графических !
;! режимах.                            !
;!                                     !
;!       В.В.Водолазкий, В.С.Головин   !
;!       03/15/91 08:58am              !
;!           (C) 1991                  !
;!-------------------------------------!
;
;
;       Графические образы мышиного курсора разработаны
;                       Е.А.Коротковой
;
MODEL   TPASCAL         ;объявление языкового интерфейса
                        ;(модели данных) Turbo Pascal
LOCALS                  ;разрешение использования локальных меток

CODESEG                 ;объявление сегмента кодов (самих процедур)

;
;   Функция проверки наличия драйвера мыши в памяти
;   Function CheckMouse : boolean;
;       возвращает TRUE, если драйвер мыши установлен,
;           FALSE - в противном случае

        PUBLIC  CHECKMOUSE
PROC    CHECKMOUSE

    PUSH    DS
    PUSHF
    CLD
    XOR     SI,SI
    MOV     DS,SI
    MOV     SI,33*4             ;читаем содержимое вектора 33h
    LODSW
    MOV     BX,AX
    LODSW
    MOV     ES,AX
    ADD     AX,BX
    CMP     AX,0
    Je      NO_MOUSE_INSTALLED
    MOV     Al,byte ptr ES:BX   ;читаем байт по адресу обработчика 33h
    CMP     AL,0CFH             ; это код iret ?
    je      no_mouse_installed
    MOV     AX,1                ;что-то установлено...
    jmp     check_mouse_ret
no_mouse_installed:
    XOR     AX,AX
check_mouse_ret:
    POPF
    POP     DS
    RET
    ENDP

;
; Начальная установка драйвера мыши и чтение текущего
; состояния
;
; Function InitMouse : boolean;
;   Возвращает  true - инициализация успешна
;               false - драйвер или аппаратура не установлены
;
        PUBLIC  INITMOUSE
Proc    INITMOUSE
    XOR     AX,AX
    INT 33                  ;функция драйвера мышки
    AND     AX,0001
    ret
ENDP
;
;   Процедура включения мышиного курсора
;   Procedure SetMouseCursorOn;
;
        PUBLIC  SETMOUSECURSORON
Proc    SETMOUSECURSORON
        MOV     AX,0001
        INT 33
        RET
ENDP

;
;   Процедура выключения мышиного курсора
;   Procedure SetMouseCursorOff;
;
        PUBLIC  SETMOUSECURSOROFF
PROC    SETMOUSECURSOROFF
    MOV     AX,0002
    INT 33
    RET
ENDP

;
;   Определение местоположения мышиного курсора и состояния
;   клавиш мыши
;   Procedure ReadMouseStatus( VAR hor,vert : word; — координаты
;                              VAR b1,b2,b3 : boolean) — кнопки
;
        PUBLIC  READMOUSESTATUS
Proc READMOUSESTATUS hor : dword,vert : dword, b1:dword,b2:dword,b3:dword
        PUSHF
    CLD
    MOV     AX,0003
    INT 33                  ;читаем состояние мышки
    LES     DI,[VERT]
    mov     ax,dx           ;вертикальная координата
    stosw
    LES     DI,[HOR]
    MOV     AX,CX           ;и горизонтальная
    STOSW
    MOV     AX,BX
    AND     AX,1
    LES     DI,[B1]
    stosb   ;левая кнопка
    MOV     AX,BX
    AND     AX,2
        MOV     CL,1
        SHR     Al,CL
        LES     DI,[B2]
        stosb                   ;правая кнопка
        MOV     AX,BX
        INC     CL
        SHR     Al,CL
        LES     DI,[B3]
        stosb                   ;средняя кнопка
```

```
        POPF

        RET
ENDP

;
;   Procedure SetMouseCursor( Hor,vert : word);
;   Установка координат мышиного курсора
;
        PUBLIC  SETMOUSECURSOR
Proc SETMOUSECURSOR   hor : word, vert : word
        MOV     CX,HOR
        MOV     DX,VERT
        MOV     AX,4
        int     33
        RET
SETMOUSECURSOR   ENDP

;
;   Procedure MouseRangeX( LeftG,RightG : word);
;   Установка диапазона перемещения курсора по горизонтали
;
        PUBLIC  MOUSERANGEX
PROC    MOUSERANGEX   leftg:word, rightg : word
        MOV     CX,[LEFTG]
        MOV     DX,[RIGHTG]
        MOV     AX,7
        INT     33
        RET
ENDP

;
;   Procedure MouseRangeY( leftG, RightG : word);
;   Установка диапазона перемещения курсора по вертикали
;

        PUBLIC  MOUSERANGEY
Proc MOUSERANGEY   leftg : word, rightg : word
        MOV     CX,[LEFTG]
        MOV     DX,[RIGHTG]
        MOV     AX,8
        INT     33
        RET
ENDP

;
;   Procedure SetGraphMouseShape( shape : word);
;   Установка параметров графического курсора
;   shape — номер картинки курсора
;
        PUBLIC  SETGRAPHMOUSESHAPE
Proc SETGRAPHMOUSESHAPE   shape : word
maxnum  equ     2               ;максимальный номер курсора
        PUSH    DS
        mov ax,  CS
        MOV     DS,AX
        MOV     ES,AX

        ASSUME DS : Code, es: code

        MOV     si,[SHAPE]
        CMP     si,0
        JLE     SH1
        CMP     si, MAXNUM
        JLE SH0
SH1:                            ;если номер курсора задан неверно
        mov si,1                ;принудительно устанавливаем первый!
SH0:    DEC si                  ;вычисляем смещение
        ;устанавливаем координаты курсора
        MOV     BX,HOT_X[SI]
        MOV     CX,HOT_Y[SI]
        ;вычисляем адрес масок курсора и экрана
        MOV     CL,6
        MOV     DX,SI
        SHL     DX,CL           ;умножаем номер на 64
        ADD     DX, OFFSET SHAPE1

        MOV     AX,9
        int     33

        pop     ds
        RET
;горизонтальная координата горячего пятна курсора
HOT_X   DW      16D,9d
;вертикальная координата горячего пятна курсора
hot_y   dw      0,8
;
;   маски курсора и экрана
;
SHAPE1:

    DW          1111111111111100B       ;маска экрана
    DW          1111111111111000B
    DW          1111111111100000B
    DW          1111111110000001B
    DW          1111111000000001B
    DW          1111100000000011B
    DW          1110000000000011B
    DW          1000000000000111B
    DW          0000000000000111B
    DW          1000000000001111B
    DW          1100000000001111B
    DW          1000000000011111B
    DW          0000001000011111B
    DW          0000011111111111B
    DW          0000111111111111B
    DW          0001111111111111B

    DW          0000000000000001B       ;маска курсора
    DW          0000000000000010B
    DW          0000000000001110B
    DW          0000000000111100B
    DW          0000000011111100B
    DW          0000001111111000B
    DW          0000111111111000B
    DW          0011111111110000B
    DW          1111111111110000B
    DW          0011111111100000B
    DW          0001111111100000B
    DW          0011111111000000B
    DW          0111100011000000B
    DW          1111000000000000B
    DW          1110000000000000B
    DW          1100000000000000B

;   второй образ — прицел

    DW          1111110000011111B
    DW          1111000000000111B
    DW          1110001000100011B
    DW          1100011000110001B
    DW          1000111000111000B
    DW          1000111000111000B
    DW          0001111111111100B
```

```
        DW          0000000010000000B
        DW          0001111111111100B
        DW          1000111000111000B
        DW          1000111000111000B
        DW          1100011000110001B
        DW          1110001000100011B
        DW          1111000000000111B
        DW          1111110000001111B
        DW          1111111111111111B

        DW          0000000111000000B
        DW          0000011010110000B
        DW          0000100010001000B
        DW          0001000010000100B
        DW          0010000010000010B
        DW          0010000010000010B
        DW          0100000000000001B
        DW          0111111000111111B
        DW          0100000000000001B
        DW          0010000010000010B
        DW          0010000010000010B
        DW          0001000010000100B
        DW          0000100010001000B
        DW          0000011010110000B
        DW          0000000111000000B
        DW          0000000000000000B



    ENDP

;
;   Procedure SaveMouseStatus;
;   Сохранение текущего состояния драйвера во внутреннем буфере
;
        PUBLIC  SAVEMOUSESTATUS

Proc SAVEMOUSESTATUS
        MOV     AX,CS
        MOV     ES,AX
        MOV     DX,OFFSET MOUSE_BUFFER
        MOV     AX,16H
        INT 33
        RET
MOUSE_BUFFER:
        db   800 dup (0)            ;двух КБ должно хватить
ENDP

;
;   procedure RestoreMouseStatus;
;   Восстановление состояния "мыши" из внутреннего буфера
;

        PUBLIC  RESTOREMOUSESTATUS

Proc RESTOREMOUSESTATUS
        PUSH    CS
        POP     ES
        MOV     DX, OFFSET MOUSE_BUFFER
        MOV     AX,17H
        int     33
        ret
        ENDP

;
;   Procedure SetMouseIntFreq(b: byte);
;   Установка частоты прерывания от "мыши"
;
        PUBLIC  SETMOUSEINTFREQ

Proc SETMOUSEINTFREQ   b : byte
        MOV     AL,[B]
        AND     AL,7
        CMP     AL,5
        JNG     SMIF1
        MOV     AL,4
SMIF1:
        XOR     AH,AH
        MOV     BX,AX
        MOV     AX,1CH
        INT     33
        RET
ENDP

;
;   Function MouseReset : boolean;
;   Программный сброс драйвера "мыши"
;
        PUBLIC  MOUSERESET
Proc MOUSERESET
        MOV     AX,21
        INT     33
        CMP     AX,21
        JE      MR1
        MOV     AL,1
        JMP     RETL1
MR1:
        XOR     AL,AL
RETL1:
        RET
ENDP

;
;   Procedure ReadMouseCounters(var X,Y : word);
;       считывание накопленного значения счетчиков "мыши"
;       с момента последнего вызова функции
;
        PUBLIC  READMOUSECOUNTERS

Proc READMOUSECOUNTERS    x : dword, y : dword
        pushf
        MOV     AX,0BH
        INT 33H
        LES DI,[Y]
        MOV     X,DX
        STOSW
        LES     DI,[X]
        MOV     AX,CX
        STOSW
        popf
        RET
        ENDP
;
;   Function LeftButtonPressed : boolean;
;   Определяет, нажата ли левая клавиша
;
        PUBLIC  LEFTBUTTONPRESSED

Proc LEFTBUTTONPRESSED
        MOV     AX,3
        INT     33                  ;читаем состояние "мыши"
        AND     BI,1
        mov     al,bl
        RET
```

```
ENDP

;
;    Function RightButtonPressed : boolean;
;    Определяет, нажата ли правая кнопка
;
        PUBLIC  RIGHTBUTTONPRESSED

Proc RIGHTBUTTONPRESSED
        MOV     AX,3
        INT     33H
        AND     BL,2
        MOV     AL,BL
        SHR     AL,1
        RET
ENDP
        END
```

Листинг 2. Интерфейс для использования ассемблерных процедур из программы на Паскале

```
Unit Grmouse;

{   В.В.Водолазкий В.С.Головин
    Модуль обслуживания позиционного манипулятора типа
    "мышь" при работе в графических режимах}

{$l mouse2.obj}
{$f+}
INTERFACE
    function  checkMouse    : boolean;
    function  initmouse: boolean;
    procedure       SetMouseCursorOn;
    Procedure       SetMouseCursorOff;
    Procedure       ReadMouseStatus( var hor,vert : word;
                                     var b1,b2,b3 : boolean);
    procedure       SetMouseCursor(hor,vert : word);
    Procedure       MouseRangeX(l,r : word);
    procedure       MouseRangeY(l,r : word);
    procedure       SetGraphMouseShape(sh : word) ;
    procedure       SaveMouseStatus;
    procedure       RestoreMouseStatus ;
    procedure       SetMouseIntFreq(b:byte);
    function        MouseReset    : boolean;
    procedure       ReadMouseCounters(VAR x,y: integer);
    function        LeftButtonPressed  : boolean;
    function        RightButtonPressed: boolean;
    function        LeftButtonReleased : boolean;
    function        RightButtonReleased: boolean;


IMPLEMENTATION
    function        checkMouse    : boolean; external;
    function        Initmouse : boolean; external;
    procedure       SetMouseCursorOn; external;
    Procedure       SetMouseCursorOff; external;
    Procedure       ReadMouseStatus( var hor,vert : word;
                                     var b1,b2,b3 : boolean); external;
    procedure       SetMouseCursor(hor,vert : word); external;
    Procedure       MouseRangeX(l,r : word); external;
    procedure       MouseRangeY(l,r : word); external;
    procedure       SetGraphMouseShape(sh : word) ; external;
    procedure       SaveMouseStatus; external;
    procedure       RestoreMouseStatus ; external;
    procedure       SetMouseIntFreq(b:byte); external;
    function        MouseReset : boolean; external;
    procedure       ReadMouseCounters(VAR x,y: integer); external;
    function        LeftButtonPressed : boolean; external;
    function        RightButtonPressed: boolean; external;

    end.
```

Листинг 3. Пример использования библиотеки GrMouse

```
program mousetest;

    uses crt,dos,graph,grmouse;


    type
        str8 = string[8];
    procedure       Print (n,m : word);
    var
        s1 : str8;
    begin
        SetFillStyle(SolidFill,Black);
        Bar(400,100,480,160);
        str(n:8,s1); OutTextXY(400,120,s1);
        str(m:8,s1); OutTextXY(400,140,s1);
    end;
{$f+}
    var
        act : byte;
        b,b1,b2,b3 : boolean; ll1,rr1 : integer;
        GraphMode,grr : integer; ll,rr : word;
    begin
        if checkMouse
            then Writeln('Драйвер мышонка в системе есть!')
            else Writeln('Мышонок в памяти хвостом не виляет...');
        b := InitMouse;
    Writeln(b);
    GraphMode := Detect;
    InitGraph(GraphMode,grr,'');
    SetGraphMode(0);
    SetFillStyle(SolidFill,Green);
    Bar(50,50,300,250);
    SetFillStyle(SolidFill,red);
    Bar(120,150,500,220);
    SetColor(14);
    MouseRangeX(0,600);
    MouseRangeY(0,400);
    SetGraphMouseShape(1);
    SetMouseCursorOn;
    b1 := false;
    repeat
    begin
    readMouseStatus(ll,rr,b1,b2,b3);
    Print(ll,rr);
    ReadMouseCounters(ll1,rr1);
            b1 := leftButtonPressed;
    if b1 then begin
        sound(50);Delay(100); NoSound;
            end;
    if RightButtonPressed then begin
        sound(2700); Delay(100); NoSound;
            end
    end
    until b1;
    SetMouseCursorOff;
    b2 := MouseReset;
    closeGraph;
    end.
```

Novintech

BORLAND

Фирма "Системы и комплексы" концерна "Новинтех" официальный представитель корпорации Borland International, предлагает программные продукты:

***** PARADOX 4.0 *****

УНИКАЛЬНАЯ ИНСТРУМЕНТАЛЬНАЯ СРЕДА ДЛЯ СОЗДАНИЯ РАЗНООБРАЗНЫХ СИСТЕМ ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ

В 10 РАЗ БЫСТРЕЕ PARADOX 3.5

"**СУБД Paradox 4.0** ориентирована на пользователей различной квалификации и предоставляет им интуитивный интерфейс в сочетании с развитой интеллектуальной средой для разработки приложений," -заявил **Г.Шапиро**, консультант фирмы MCA. -"Приспособляемость к квалификации пользователей, а также создание условий для ее повышения, расширение масштабов приложений по мере возрастания практических потребностей пользователей и предприятий - вот те основные причины, по которым **СУБД Paradox 4.0** стала нашим стандартом."

"**СУБД Paradox 4.0** - это мощная система управления базами данных, которая несомненно будет выбрана многими предприятиями в качестве стандарта," - сказал **Филипп Кан**, президент фирмы **BORLAND**. " Наша фирма в настоящее время предоставляет предприятиям, планирующим и в будущем работать в различных операционных средах, эффективное решение многих оперативных задач. **СУБД Paradox 4.0** подтверждает стремление фирмы **BORLAND** создать для широкого круга пользователей такую систему, которая бы в максимальной степени отвечала их практическим потребностям."

В среднем **СУБД Paradox 4.0** в **десять раз** превышает быстродействие **Paradox 3.5** и намного опередила известные в настоящие время СУБД по этому показателю, в том числе и **FoxPro/LAN 2.0**. **СУБД Paradox 4.0** функционирует на ПЭВМ типа IBM PC (на базе Intel 80286 и Intel 80386) с оперативной памятью не менее 2 мгб. **СУБД Paradox 4.0** является первой системой, в которой обеспечена полная поддержка интерфейса **DPMI**, использующего механизмы управления памятью, реализованные в операционной среде **Windows 3.0** или выше. **СУБД Paradox 4.0** использует расширенную память и функционирует в защищенном режиме под управлением **Windows**.

- Язык программирования **PAL** (Paradox Application Language) включает ряд новых возможностей, позволяющий пользователям создавать разнообразные приложения, использующие процедуры управления "мышью", "нисходящие" или "выпадающие" меню, отвечающие современным требованиям, диалоговые панели с программными клавишами, контрольные панели и т.п. **Application Workshop** - это развитый генератор приложений, который дает возможность пользователям в диалоговом режиме создавать приложения, отвечающие их конкретным потребностям. **Paradox Engine**, реализованный в виде библиотеки объектных модулей, обеспечивает интерфейс к **СУБД Paradox 4.0** для прикладных программ, написанных на языке программирования **C** и **Pascal**. Прикладная программа, использующая функции из этой библиотеки, может манипулировать таблицами **Paradox 4.0** как в однопользовательском, так и в многопользовательском режимах. Результатом создания прикладной программы является стандартный выполняемый файл (.EXE), с присущими ему достоинствами независимых программ.

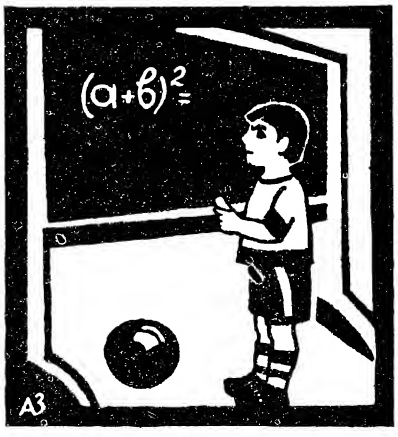
СУБД Paradox 4.0 предназначена для применения как в среде операционной системы **MS-DOS**, так и **MS-Windows**. Для пользователей, работающих только в **MS-DOS** или смешанной среде **MS-DOS/MS-Windows**, **СУБД Paradox 4.0** представляется значительным достижением в области систем управления базами данных на ПЭВМ. Она может удовлетворить самые взыскательные требования к реакции системы на сложные запросы к базе данных, скорости обработки информации в сетевой среде, набору типов данных (в частности, возможности хранения сложных типов данных - *графических изображений*, *аудио/видео информации* и *других двоичных данных* - в **MEMO**-полях) и механизмам управления памятью, обеспечивающим защищенный режим выполнения операций.

Для предприятий, которым требуется обеспечить доступ к информации, размещенной на удаленном **SQL-сервере**, **СУБД Paradox 4.0** совместно с модулем **Paradox SQL Link** предоставляет возможности для безопасного, прозрачного и высоко эффективного доступа к различным **SQL-серверам**. Кроме того, пользователи получают большую гибкость при работе с информацией, хранящейся в удаленных файловых серверах. В обоих случаях пользователи получают все преимущества **СУБД Paradox 4.0**, не прибегая к изучению нового языка или интерфейса.

В **СУБД Paradox 4.0** реализованы эффективные способы доступа ко всем известным в настоящее время системам управления базами данных, включая **ORACLE 6.0**, **Microsoft и SYBASE SQL Server**, **IBM OS/2 Database Manager**, **DB2 через MDI Database Gateway** и **VAX Rdb/VMS 4.1 (DEC)**. С сентября 1992 г. пользователи смогут получить модуль **Paradox SQL Link**, обеспечивающий взаимодействие с **Novell NetWare SQL 3.0**.

Пользователи и разработчики информационных систем с большим удовлетворением найдут в **СУБД Paradox 4.0** *много новых и удивительных возможностей, о которых им приходилось лишь мечтать, пользуясь программными продуктами других фирм.*

Фирма "Системы и комплексы"
концерна "НОВИНТЕХ"
Адрес: 119517, Москва, ул.Нежинская, 13
Тел. (095) 441-11-00
Факс (095) 442-57-92



Вычисляем по формуле ...

М. Черкашин

Процедура вычисления выражения будет вам полезна в любой программе.

При всем разнообразии существующих языков программирования одна конструкция есть, кажется, во всех. Это выражение. Когда требуется что-то вычислить, имеет смысл задать это "вычисление" с помощью явной формулы. Например, так:

Вот это и называется выражением. Кстати, оно требуется далеко не только в языках программирования.

Хотя нам и ясно, что мы называем выражением, все-

```
5 * (6 + x)
7 + 8 * 36 - l
5^x + 5^y
```

таки следует знать это чуть-чуть лучше - ведь придется писать программу для его вычисления. Так что же такое на самом деле выражение?

Несомненно число или переменная - это выражение. Если взять два выражения и соединить их знаком операции, например, + или #, тоже получится выражение. Выражение, заключенное в скобки, также будет выражением. То, что мы сейчас записали, можно сказать и короче. Обозначив выражение буквой E и заменив стрелкой слово "получится", имеем:

```
E -> (E)
E -> E S E
E -> число
E -> имя
S -> +
S -> *
S -> ^
S -> -
S -> 1
```

E — выражение
S — знак операции

Этот способ обозначения называется формой Бэкуса - Наура. К сожалению, такое определение выражения нас не устраивает. Рассмотрим, например, такое выражение:

```
l + 6 * d
```

Из нашего определения никак не следует, что сначала делается умножение, а затем сложение. А хотелось бы, чтобы при разложении выражения на части получались выражения поменьше, которые мы действительно получаем при вычислении "большого" выражения, а не что-то непонятное , не имеющее к нему никакого отношения. Поэтому нам придется изменить определение выражения. Так, мы должны объявить, что если получили выражение с помощью сложения, то мы не можем его сразу на что-то умножать - иначе получим совсем не то, что хотели. Другими словами, умножать мы можем выражения, полученные с помощью умножения же, или возведения в степень, или заключенные в скобки. А теперь посмотрим "исправленное" определение выражения.

```
E -> L + E   L -> T * L   T -> N^T   N -> имя
E -> L - E   L -> T / L   T -> N     N -> число
E -> L       L -> T                  N -> (E)
```

E — выражение
L — слагаемое
T — множитель
N — основа

В отличие от первого определения, второе имеет еще одну полезную особенность. С его помощью можно собрать нужные нам выражения из переменных, чисел и операций единственным способом.

На рис. 1 приведен пример такого выражения. Предположим, нам необходимо его вычислить. Для этого следует произвести некоторые действия. Иногда мы можем менять порядок, в котором их производим, иногда нет. Например, не ясно, что следует делать вначале - делить на четыре число k или возводить d в куб. Однако вычитать мы всегда должны в последнюю очередь.

Более наглядно: нанесем на чистый лист бумаги все операции, переменные и константы, использованные нами в выражении. И от операций проводим стрелки к значениям (или к другим операциям), если они используются той операцией для вычисления. В целом получается конструкция, похожая на генеалогическое древо. Только вместо потомков - выражения, используемые данной операцией.

А теперь самое главное - как это выражение вычислять. Проиллюстрируем наши рассуждения примером програм-

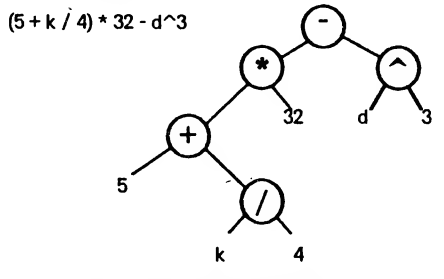


Рис. 1. Представление выражения в виде "дерева"

мы. Добавив сканер (рассмотренный в предыдущей статье), получим готовую программу, которую можно использовать вместо микрокалькулятора. Вычисление выражения состоит в основном из трех этапов. Считывание и вычисление первого операнда. Считывание операции. Считывание и вычисление второго операнда. К сожалению, здесь есть маленькая трудность - как определить, какой операнд к

какой операции относится. Ведь операнд бывает и переменной, и целым выражением. Конечно мы можем постепенно вычислять выражение, начиная с тех операций, которые могут быть вычислены первыми, но это заведомый садизм. Хотелось бы вычислить выражение, не "пробегая" по нему несколько раз.

Один из алгоритмов работает по следующему принципу: просматривает выражение слева направо, вычисляет то, что может, а все, что пока нельзя использовать, запоминает.

Допустим, выражение имеет вид, изображенный на рис. 2.

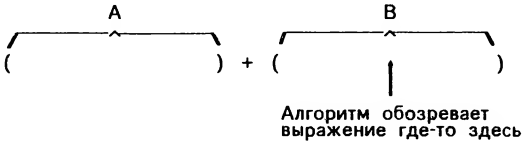


Рис. 2. Структура вычисляемого выражения

К текущему моменту алгоритм уже вычислил подвыражение A и сейчас занимается подвыражением B. Поэтому алгоритм должен хранить в себе значение выражения A (уже вычисленное) и операцию "+". Эта информация понадобится алгоритму, когда вычисление подвыражения B будет полностью завершено. Если же подвыражение B само состоит из подвыражений, при его вычислении может понадобиться сохранять в памяти еще какую-то информацию. К нашему счастью, когда нам понадобится значение выражения A, эта промежуточная информация будет уже не нужна. Поэтому при вычислении выражения используется структура данных, называемая "стек". Эта структура данных - как рожок автомата: вычислили, но почему-то не можем сразу использовать - кладем в стек. Потом поверх него лягут другие порции информации - и только после того, как они будут использованы, дойдет очередь и до этой.

Посмотрим на листинг. Процедура Compute для каждой лексемы (переменной, символа операции или скобки) означает положить в стек или попытаться вычислить.

Обратим внимание на процедуру ComTo. Иногда до конца выражения или до следующей за подвыражение B операции нельзя сказать, здесь ли кончается выражение. Тогда из стека приходится вытаскивать сразу по несколько значений. Мы это называем "довычислить" выражение. Посмотрим на рис. 3. Там показан пример, как для того чтобы определить конец подвыражения, иногда нужно заглянуть чуть-чуть дальше.

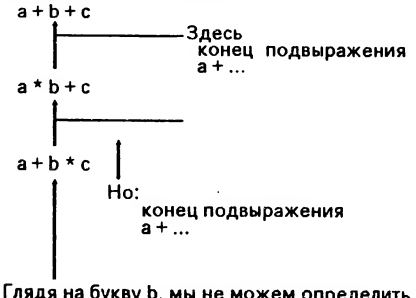


Рис. 3. Выделение подвыражений

Процедуры Push и Pop – положить в стек и достать из стека.

Э к з о т и к а. Обычно выражения в языках программирования мало отличаются от привычного вида. Меняется набор операций, их приоритеты – и все. Но бывают и исключения.

Традиционный способ записи выражения - это так называемый инфиксный способ записи, когда операция ставится между двумя операндами. Но можно поступать и по-другому. Например, сначала написать оба операнда, а уже потом операцию. Или наоборот - сначала операцию, а потом оба операнда. Первый способ называется постфиксной польской формой записи, а второй - префиксной.

Например,

(5 + k / 4) * 32 - d^3

Инфиксная

5 k 4 / + 32 * d^3 -

Постфиксная

- * + 5 / k 4 32 ^d3

Префиксная

У этих форм есть свои достоинства. Например, не нужны скобки: каждая операция просто забирает столько операндов, сколько ей нужно. Затем вычислить значение по этим формам записи значительно проще. Особенно по постфиксной (префиксную можно начинать вычислять с конца, тогда получится постфиксная). Достаточно одного стека. Встречаешь число или переменную - отправляешь в стек, встречаешь операцию - берешь нужное число операндов из стека, вычисляешь и результат опять кладешь в стек. Поэтому, если выражение придется задавать лишь изредка, подумайте - может, стоит сделать префиксную форму записи.

А вот как выглядят выражения в языках Лисп и Форт. Как видим, в Лиспе используется префиксная, а в Форте постфиксная форма записи:

$b^2 - 4ac$

А на Лиспе:
(- (*bb) (* 4 (*ac))
А на Форте:
b b * 4 a c * * -

Листинг. Вычисление выражения.

```
Type  Lexems = (LexNumber, LexOpen, LexClose, LexOp);
     Ops = (OpParn, OpPlus, OpMin, OpMul, OpDiv, OpPower);
Var Sta, Stream : array [1..Many] of integer;
     OpTop, StTop : integer;

procedure Compute;
begin
     while not Finish do
     begin
          Scan;
          case LexType of
               LexNumber     : Rpush (LexVal);
               LexOpen       : Push (OpParn);
               LexClose      : ComTo;
               LexOp         : while Prty (Sta [OpTop]) >=
                                    Prty (LexVal) do ExecOp;
            else WriteLn ('Неверная лексема');
          end;
     end;
end;

procedure ComTo;
begin
     while   not (OpTop = 0) and
             not (Sta [OpTop] = OpParn)
          do ExecOp;
end;

procedure ExecOp;
var x,y,z : integer;
begin
     y := Rpop;
     x := Rpop;
     case Pop Of
          OpPlus : z := x+y;
          OpMin : z := x-y;
          OpMul : z := x*y;
          OpDiv : z := x/y;
          OpPower: begin
               z := 1;
               for i:=1 to y do
                    z := z*x;
             end
          else WriteLn ('Такой операции нет!');
end;
```

PC_Multi — инструмент для профессионалов

Первое поколение прикладного математического обеспечения на IBM PC представляло собой как правило однопользовательские монстры типа "АРМ парткома" или "САПР бюрократа". Но вот прошло несколько лет, машины стали доступнее и сразу же обнаружилось, что основной класс задач связан с удаленной обработкой и/или сбором информации и обеспечением совместной работы нескольких пользователей. В результате на наш рынок хлынул поток локальных сетей: Novell, LANtastic, LANSmart и т.д.

Но всегда ли нужна эта тяжелая артиллерия, применение которой выливается в серьезные затраты на аппаратное и программное обеспечение, а в результате - три четверти объединенных в сеть ЭВМ используются в качестве пишущих машинок? Использование PC_Multi позволяет организовать одновременную работу нескольких пользователей на одной ЭВМ (например, восемь пользователей АРМ "Склад"), предоставив каждому свой терминал. В качестве терминалов могут использоваться как современные ZELAX (Зеленоград), поставляемые фирмой Альтер, так и старые дисплеи фрязинского завода, пылящиеся возле забытого в углу ДВК-2.

Инструментальная система PC_Multi, предназначенная для создания информационных и управляющих систем на языке Си, разработана фирмой Альтер, разместившейся в живописном (и ароматном - на территории кондитерской фабрики) уголке Москвы: Ленинградский проспект, 15. Взаимодействие прикладных программ со средствами PC_Multi обеспечивается использованием объектных библиотек.

В состав PC_Multi входят следующие компоненты.

1) Мультитерминальная система ввода/вывода (МТС).

Предназначена для обеспечения поддержки до 8 дисплеев в режиме локального терминала. Система содержит драйвер мультиплексора, непосредственно обслуживающий дисплей (система команд VT52), и набор библиотек поддержки пользовательского интерфейса (окна, меню, формы ввода, горячие клавиши, подсказку).

Если Вы поначалу не используете несколько терминалов, но использовали при написании программ библиотеки поддержки МТС - это означает, что, подключив в любой момент дополнительный терминал, Вы сможете использовать его в работе, не меняя ни одной строки программы.

2) Многозадачный монитор реального времени.

Обеспечивает выполнение нескольких программ в среде MS DOS в режиме реального времени.

Переключение задач осуществляется по прерываниям от системного таймера, что определяет минимальный квант времени, выделяемый процессу, в 55 ms.

Поскольку достаточно часто пользователи, работающие за соседними терминалами, применяют одну и ту же программу (например, работают с базой данных), то расточительно одну и ту же программу копировать в оперативной памяти ЭВМ по числу подключенных терминалов. Монитор дает возможность загрузить программу в ОЗУ только один раз, обеспечивая смену контекста задачи (данные, регистры) при переключении процессов. Это достигается путем специальной сборки программ, предназначенных для реентерабельного использования.

3) Система доступа к данным.

Любая информационная система - это прежде всего структура данных и ее обслуживание. Система поддержки данных, реализованная в PC_Multi, позволит Вам использовать базы данных, созданные с помощью СУБД Paradox и DBase III+.

Такой подход, в противовес разработке каких-то уникальных структур представления данных, позволяет Вам в полной мере использовать уже созданное математическое обеспечение, не отвлекая силы на переписывание старых, работающих программ, и кроме того позволяет использовать СУБД при создании макетов баз данных.

4) Локальная сеть Ulter–Net.

Представляет собой одноранговую локальную вычислительную сеть, сопоставимую по возможностям с LANtastic или LANSmart. Ulter_Net ориентирована на использование коммуникационных портов RS-232C (скорость передачи данных до 115 КБод). Проигрывая сетям, ориентированным на высокоскоростные (и дорогие!) адаптеры по скорости передачи данных, Ulter_Net выигрывает по объему занимаемой памяти. Сетевое программное обеспечение на рабочей станции занимает около 12 КБ, на сервере - 8 КБ, на узле, совмещающем рабочую станцию и сервер - 16 КБ. Сеть Ulter_Net "уживается" с матобеспечением фирмы Novell и обеспечивает доступ к серверу этой сети через рабочую станцию Novell.

Любой программный продукт предъявляет вполне определенные требования к составу аппаратных средств. PC_Multi отличается достаточной гибкостью и возможностью использования широкой гаммы вычислительных платформ и периферийных устройств. Но если Вас беспокоит совместимость, Альтер поставит Вам:

- - платы 8-канальных мультиплексоров (реализованных по Вашему выбору на i8250 или на 580ВВ51);
- - терминалы;
- - компьютеры.

Оплата производится (внимание, граждане стран СНГ, вышедшие из "рублевой зоны!) в рублях.

Фирма Альтер готова доработать математическое обеспечение под любую имеющуюся в распоряжении заказчика аппаратуру. И конечно фирма поставляет информационно-вычислительные системы "под ключ" в соответствии с требованиями заказчика.

Товарищество с Ограниченной Ответственностью «АЛЬТЕР»
Адрес для переписки: 125140. Москва, Ленинградский проспект 15, тел. (095) 251-29-46
Расчётный счёт № 1467615 в Межрегионбанке города Москвы, МФО 211899, уч. ВЦ-С2
ULTER

SYMANTEC

The software side of life

Десять лет истории,
Десять сотен сотрудников,
Десять миллионов пользователей,
И одна идея – Создавать лучшие программы в мире.
Symantec

Zortech C++

For Windows, DOS & OS/2

Профессиональные многоплатформные средства для создания превосходных прикладных программ

Zortech C++ - один из наиболее мощных инструментов создания прикладных программ, имеющихся сегодня на мировом компьютерном рынке.
Zortech имеет репутацию элитарного продукта, предназначенного для истинных профессионалов программирования. И мы хотели бы поддерживать эту традицию; нам, безусловно, лестно осознавать, что пользователями Zortech являются люди, направляющие развитие компьютерной индустрии.
Мы полагаем, что Zortech будет весьма полезен для программистов стран Содружества и поможет им полнее раскрыть свой потенциал, предлагая средства для создания программных продуктов, отвечающих высоким критериям качества и тенденциям в развитии мирового рынка.

Одна из основных тенденций - создание прикладных программ, имеющих аналоги в нескольких платформах, например: Time Line для DOS - Time Line для Windows; Norton BackUp для DOS - Norton BackUp для Windows, Lotus 123 для DOS - Lotus 123 для Windows и так далее. Эти программы, несмотря на то, что они созданы для разных платформ, могут (или будут) свободно обмениваться данными, что создает огромные возможности для распространения информации между пользователями, использующими различные системы. А свободный обмен данными, в свою очередь, это не только направление в компьютерной индустрии, но и ее девиз на 90 годы.
Семейство программных продуктов Zortech позволит Вам создавать прикладные программы для DOS, Windows, OS/2, UNIX и Macintosh.

В чем заключаются достоинства Zortech ?
Прежде всего в том, что он генерирует компактные, сопоставимые по размеру с C, 16-и и 32-х разрядные коды на C++ для компьютеров с 8086/8088, 80286, 80386 и 80486 процессорами. Zortech может быть использован на компьютерах AT и даже XT. Администратор кодов виртуальной памяти (Virtual Code Manager) позволяет легко создавать сверх-большие программы и обрабатывать сверх-большие объемы данных. Zortech предлагает полный набор абстрактных типов данных (АТД) и хэш-файлы , классы для страничной организации памяти. Zortech имеет набор утилит, часть из которых поставляется в исходных кодах, облегчающих разработку ООП. Математика с плавающей запятой Zortech (M++) не знает себе равных . Zortech имеет библиотеки WINC для конвертации программ для MS DOS в приложения под Windows. В комплект поставки входит документация на Windows SDK в печатном виде.

За более подробной информацией о семействе программных продуктов Zortech обращайтесь к дистрибуторам

Symantec:

115409, Москворечье, 31/2 АО
"Диалог–МИФИ", тел 320–3466,
факс 324–3055

Merisel, 109044, Крутицкий вал, 3,
корп. 2 тел 276–4714,
факс 274–0097

Zortech - зарегистрированная торговая марка Symantec Corporation MS DOS, Windows - зарегистрированные торговые марки Microsoft Corporation Lotus 123 - зарегистрированная торговая марка Lotus Development Corporation

Некоторые предпочитают Zortech

А. Сидоров

В настоящее время большинство программистов работает на персональных ЭВМ в операционных системах типа DOS, разрабатывая программное обеспечение (ПО) для DOS и Windows. Но наблюдается устойчивая тенденция роста разработок ПО для Windows, Unix и различных локальных и глобальных сетей . По-прежнему пользуются спросом CAD/CAM системы , программы обработки информации, поступающей от аналоговых устройств (сканеров, теле- и фотокамер, различных датчиков и т.д.), а также новое и очень перспективное направление, - мультимедиа (multimedia).

Функционирование подобных систем возможно только при условии полного использования ресурсов "железа" и операционных систем . Для обработки огромных информационных массивов в реальном режиме времени (чтобы пользователь не заскучал), достигая при этом уровня визуализации WYSIWYG на всех устройствах вывода и обладая развитым пользовательским интерфейсом (ПИ), максимально отвечающим стандартам SAA/CUA, такие разработки представляют сложные программные системы с повышенными требованиями к качеству и эффективности. Разработка надежного системного ПО большого объема требует использования высокоэффективных инструментальных средств, в первую очередь - систем программирования (СП).

Естественно только на ассемблере этого уже не сделаешь,- нужны СП более высокого уровня.

При разработке больших программных комплексов ("больших" - в смысле объемов и сложности связей: около 50 000 - 100 000 строк исходного кода) возникает ситуация, когда один разработчик уже не в состоянии помнить (отслеживать) все связи и соглашения по комплексу (компоненте). А механические добавления количества разработчиков отнюдь не облегчают задачу, а даже наоборот - усложняют ее. Даже несмотря на самые совершенные спецификации, эта задача остается весьма сложной. Существует множество методологий в разработке ПО, но ключевым вопросом остается один: каких принципов следует придерживаться при разработке ПО и систем программирования?

В 90-е годы это будут объектно-ориентированный подход к программированию (ООП) и CASE (автоматизированные средства разработки ПО). А признанным лидером среди объектно-ориентированных языков является C++, и интерес к нему все возрастает, как к средству, позволяющему одному программисту создавать коды в 25-50 тыс. строк без потерь эффективности и надежности, легко переносимые на другие процессоры, ОС и операционные оболочки типа MS Windows, PM OS/2 или X-Windows.

Еще летом 1988 года Zortech C++ был первым полноценным компилятором на IBM PC, который генерировал код для процессоров 80x86. Zortech с 1988 года выпустил 7 версий компилятора C++ от первого транслятора, генерирующего код для Intel 8086, MS Windows и OS/2, до компилятора, который позволяет создавать программы с виртуализацией памяти для 32-битовых процессоров 386/486 в DOS, OS/2 и SCO UNIX.

Сегодня есть из чего выбрать ! Только за последние 2-3 года разработаны около десятка компиляторов C++: от примитивнейших препроцессоров Си до промышленных оптимизирующих компиляторов, почти полностью перекрывающих стандарт cfront. Некоторые из этих компиляторов состоят только из строковых трансляторов и небольших библиотек , другие представляют мощнейшие системы программирования с большими наборами библиотек классов и инструментальных средств, поддерживающих CASE технологию. К сожалению, часто бывает так , что первые имеют отличные по качеству компиляторы, а вторые - отличные средства разработки и обучения , но весьма посредственные компиляторы с повышенным содержанием ошибок типа "Зря ты это затеял, друг ...".

Парадоксально, что к числу компиляторов второго типа относятся последние разработки монстров индустрии программного обеспечения – Borland C++ 3.0 и MicroSoft C/C++ 7.0. Последние два года Borland наперегонки (с кем ?) штамповал с фантастической скоростью новые ООП - трансляторы, Microsoft стоял в стороне и говорил: "Придет время и я все скажу, что думаю по этому поводу...".

Время пришло - с MS C/C++ 7.0 нельзя работать из-за фантастического количества ошибок и недоработок, весьма странных для такой солидной фирмы. Видимо у Гейтса не выдержали нервы, и он выпустил раньше срока недоведенный продукт, или все силы ушли на Windows 3.1 (действительно - вещь!). Так или иначе , но пока MS C/C++ 7.0 в

полном объеме лучше не приобретать и подождать доведенной версии: какой-нибудь MS C/C++ 7.1. А деньги потратить на улучшение "железа": на приобретение машины с 386/486 процессором, расширение памяти как минимум до 4M ... MS C/C++ 7.0 просто не работает хотя бы без 80386SX с 4 мегабайтами памяти.

У Borland C++ 3.0 требования на аппаратуру поменьше: хватает 80286 , но 2M оперативной памяти минимум и до 42 мегабайтов на диске. Но качество исполнения не намного выше , чем у своего заклятого друга. Тут я впервые увидел , как может вылететь по защите памяти строковый компилятор(!) на трансляции 40-строчной программы при 3 свободных мегабайтах ! Конечно, даже несмотря на все ошибки, среда разработки, новые функции и классы, попытка введения template в язык оставляют очень благоприятное впечатление. Object Windows и Application Framework - просто великолепны ! Но...пока лучше воздержаться от покупки Borland C++ 3.0 . Подождем версию 3.x, где не будет такого количества ошибок, появится поддержка Windows 3.1 и наконец, может быть будет нормальная арифметика с плавающей точкой !

Повышенный интерес в последнее время вызвал Watcom: в первую очередь из-за реализации на нем Foxbase и Novell. Но после приобретения Fox очень сомнительно, что Microsoft допустит реализацию своего продукта на компиляторе другой фирмы.

Семейство TopSpeed компиляторов фирмы JPI тоже весьма популярно, но в последнее время сдало свои позиции. Да и количество ошибок по-прежнему слишком велико.

Из всех этих продуктов я предпочитаю Zortech C++ 3.0. Zortech C++ 3.0 существует в двух редакциях:

- Zortech C++;
- Zortech C++ Science & Engineering Edition (версия для научных и инженерных приложений).

Эти редакции отличаются разным наполнением библиотек и инструментальными средствами в зависимости от того рода задач, для которых они предназначены. Рассмотрим их подробнее.

Математика Zortech C++ отличается повышенной скоростью и малыми размерами реализации, что представляет интерес для всех, кто разрабатывает математические,статистические и инженерные приложения, а также обработку графических образов.

Компилятор Zortech C++ создает более быстрый код за счет уникальной многоступенчатой глобальной оптимизации.(На 20-40% быстрее, чем Borland C++!). По размеру создаваемого кода Zortech C++ - вне конкуренции !

В математической и инженерной версиях у Zortech C++ есть великолепное средство для разработки математических приложений - M++. Это мощное средство абстракции и моделирования математических программ. M++ это полный, стандартизированный язык многомерных массивов. Используя C++ и простейшие расширения M++, Вы можете просто работать с многомерными массивами (матрицами) так же, как и на бумаге. M++ более мощный, чем Фортран, и несомненно более интеллектуальный. При этом математика отвечает стандарту IEEE-754-1985. Численное программирование на M++ доставляет удовольствие.

Особенно полезен MM++ для высокоточных расчетов математического и научного характера, для инженеров и ученых (особенно для тех, кто использовал и Фортран).

Zortech C++ предоставляет уникальную возможность - интерфейс с Фортраном, т.е. позволяет присоединять модули на Фортране к программе на C/C++! Достаточно лишь добавить ключевое слово fortran к прототипу функции.

Zortech C++ позволяет использовать стандартные операции над матрицами, тензорами с любой точностью, оставляя возможность подключать программы на Фортране для любителей. Это позволит создавать и переводить мощные наработки в области математики, статистики, математической физики и т.д. на совершенно другой уровень исполнения. При этом переносимость программ остается прежней и даже возрастает везде, где идет C++, пойдут и Ваши математические приложения!

Только в Zortech C++ можно написать :

```
double MyArray[10000]; // 8*10000 = 80000 байт >> 64K !
```

и больше ни о чем не заботиться! Главное,чтобы хватило места на диске - виртуализация происходит автоматически - об этом заботится VCM - Virtual Code Manager. Отпадает масса проблем, связанных с распределением оверлейных буферов, и т.п.

Zortech C++ предоставляет возможности для разработки приложений на мощных машинах. Это расширения DOS для 386/486 процессоров Phar Lap 386 и Rational System 286, а также возможность разработок под OS/2 и SCO UNIX. Только Zortech C++ поддерживает сопроцессор Weitek!

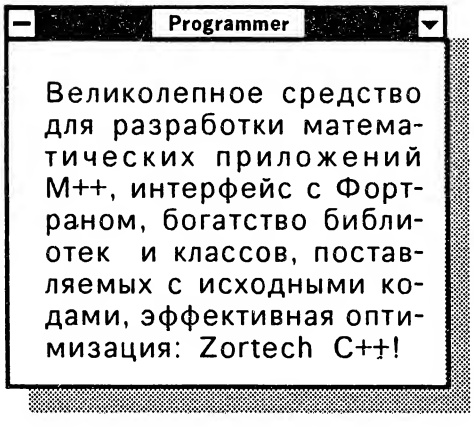
В состав системы входит ZPM - бесплатное расширение DOS для 286 и DOSX 32-битовое расширение для 386 DOS.

Графика Zortech C++ - Flash Graphics (FG) поддерживает CGA, EGA, VGA, VESA 800x600, MCGA, Hercules, Toshiba 3100, IBM 8514 работает и в 386 защищенном режиме и в OS/2 (без PM).

Flash Graphics более мощная и быстрая, чем BGI Borland. Это следствие более быстрой и компактной математики с плавающей точкой Zortech.

Zortech C++ может создавать код в DOS для 286,386 и 486 процессоров, поддерживающий расширенную память. Есть кодогенератор для OS/2 и XENIX.

Сегментный компоновщик Zortech BLINK совместим с MS Link, но имеет и массу отличий , позволяющий ему создавать уникальные по качеству программы. В отличие от обычных компоновщиков BLINK по умолчанию дополняет все программы плавающей точкой.



Одна из "изюминок" Zortech C++ - это возможность генерировать программы для процессоров 80x86, выполняемые без участия ОС, т.е. - со своим загрузчиком! Это открывает широкое поле деятельности для истинных хэкеров и любителей писать свои операционные системы!

Zortech C++ может создавать 8 типов EXE файлов:

- обычные MS-DOS;
- для расширений ДОС: Rational 286,ZPM 286, Pharlap 386,DOSX 386;
- для MS Windows 3.0;
- для OS/2 1.x;
- для SCO Unix!

Нужно помнить, что существует версия Zortech C++ и для Macintosh. Таким образом сильно облегчается перенос ПО с IBM на Mac. Мобильность и многоплатформность - одна из основных черт программных продуктов Symantec.

Очень важное значение имеет оптимизация: особенно для C++, так как inline функции и ссылки производят слишком большой код. Глобальный оптимизатор Zortech C++ разрабатывался и совершенствовался более 8 лет. Очень эффективен. В отличие от других оптимизаторов дает гарантию, что оптимизированная программа будет работать по-прежнему, в том числе и в разработках под MS Windows.

Великолепное впечатление оставляют библиотеки Zortech. Все они поставляются с исходными кодами, что дает широкий простор для разработчиков по модификации и переносимости приложений на C++ в другие системы. Богатству библиотек и классов Zortech C++ нет аналога среди компиляторов C/C++ для PC. В состав инструментальных классов входят следующие группы:

битовые векторы и множества;
списки, очереди,стеки;
двунаправленные списки;
динамические массивы;
бинарные деревья;
хэш таблицы;
хэш файлы;
bcd арифметика;
виртуальные массивы;
время и дата (обработка);
часы;
обработка списков директорий;
имена файлов;
обработка вектора прерывания;
обработка ошибок dos;
редактор строк;
текстовые окна;
редактор текстов;
специальный класс для денежных расчетов ;
очередь событий;
пользовательский интерфейс: экран, интерактивные объекты, кнопки, меню, подсказки и т.д.

Кроме инструментальных классов для C++ есть весьма богатый набор библиотек для Си, включающий в себя пакеты для работы с BIOS и DOS, экраном и принтером, EMS и виртуальной памятью, обработки прерываний и создания TSR программ, страничным обменом и своппингом, таймером и звуком, не считая графики, списков (a la Лисп), работой с именами файлов и файловой системы, меню и функций обработки строк.

Используя Zortech C++ можно буквально в считанные часы создать прототип системы в соответствии с принятыми сейчас стандартами на пользовательский интерфейс. Располагая уже готовыми классами задач, не составит большого труда создать серьезные прикладные системы под DOS, Windows, OS/2 и UNIX.

Zortech C++ предназначен для профессионала, который знает в достаточном объеме C/C++ и операционные системы. Zortech C++ один из немногих компиляторов, для которого в документации на диске приводится список уже обнаруженных ошибок и рекомендации для их обхода. Недостатком Zortech C++ являются слабая (по сравнению с Borland C++) среда разработки и отсутствие таких средств как ассемблер и профайлер. Но эти недостатки с лихвой окупаются мощным оптимизирующим компилятором, богатейшими библиотеками, средствами поддержки объектно-ориентированного программирования и расширениями для математических приложений.

Можно порекомендовать Zortech C++ как средство для создания систем в областях, связанных с большими математическими вычислениями. Это могут быть пакеты статистической обработки данных, принятия решений, численных методов, математической физики, астрономии, систем автоматизированного проектирования, обработки графических образов и там, где требуются высокая скорость вычислений, большой объем обрабатываемой информации и возможность использования имеющихся пакетов на Фортране и C, где нужна переносимость приложений с персонального компьютера на более мощные вычислительные системы.

SUNRISE

Продаем оптом и в розницу
за наличный и безналичный расчет
компьютеры любой конфигурации от PC XT до PC AT 486/487.
Возможен бартер!

Наши телефоны: 408-83-54, 485-44-81.

Работа с DXF файлами в Автокаде 11

Р. Глуховский

База данных рисунка Автокада (файл с расширением .dwg) хранится в сжатом формате, который подвергается изменениям от версии к версии по мере добавления в Автокад новых функциональных возможностей, поэтому документация на этот формат не публикуется. Мы не рекомендуем создавать программы, использующие формат файла .DWG.

Для того чтобы многофункциональный графический редактор Автокад мог обмениваться информацией с внешним миром, фирма Autodesk изобрела формат файла обмена рисунками DXF (Drawing Exchange Format). Наличие такого формата дает возможность пользователям Автокада импортировать рисунки, созданные другими графическими редакторами, а пользователям других САПР – импортировать рисунки Автокада. Кроме того, с помощью DXF файла, который является по существу стандартом, можно обмениваться информацией с внешними расчетными программами. Это означает, что геометрическую информацию о модели можно передать из Автокада на мощную большую машину, провести структурный анализ методом конечных элементов и передать обратно в Автокад данные для отображения деформированной модели. Формат DXF воспринимается всеми версиями Автокада и может быть преобразован во внутреннее представление файла рисунка (и наоборот). DXF файлы - это обычные текстовые файлы в коде ASCII. Они могут легко преобразовываться в форматы других систем автоматизированного проектирования и передаваться во внешние (по отношению к Автокаду) прикладные программы для специальной обработки.

DXF файл содержит информацию, необходимую для воссоздания рисунка Автокада. Этот файл является основным источником информации, поскольку структура первоисточника, т.е. DWG файла, не документируется. К сожалению, в DXF файле отсутствует один важный элемент - это описание используемых форм и шрифтов. Они хранятся в отдельных файлах (SHX).

DXF файл состоит из четырех изолированных разделов. Если DXF файл создается пользователем, он может включать только разделы, содержащие необходимую информацию о конструируемом рисунке. Если же этот файл создается Автокадом, в нем можно найти все четыре раздела, и поэтому следует знать их структуру, а именно:

HEADER - заголовок;

TABLES - таблицы;

BLOCKS - блоки;

ENTITIES - примитивы.

HEADER

Данный раздел содержит имена и значения переменных среды рисунка на момент создания DXF файла. (Полный список этих переменных можно найти в Приложении А к Руководству пользователя Автокада.) Итак, в заголовке можно найти:

- версию Автокада, с помощью которой создавался рисунок;
- координаты пределов рисунка;
- координаты границ рисунка;
- параметры размерных переменных;
- имя текущего меню;
- текущие значения всех системных переменных.

Другая важная информация в заголовке - это запись о номере версии. Эта запись является ключом к тому, какие другие значения присутствуют в DXF файле. Например, программа, преобразующая последнюю версию DXF файла к более ранней (чтобы осуществить совместимость с предыдущей версией Автокада), должна определить версию, в которой создавался рисунок, и проигнорировать все значения и элементы, недоступные предыдущей версии.

Использовать эту информацию или нет, зависит от Вашего программного приложения. Например, для преобразования Вашего рисунка в файл на языке фотоплоттера и вычисления масштабных коэффициентов полезны значения пределов и границ рисунка, остальную же информацию можно проигнорировать.

TABLES

В отличие от раздела HEADER, данный раздел содержит информацию о группах более сложных рабочих параметров рисунка Автокада и имеет восемь подразделов.

Подразделы раздела TABLES

APPID - используемые приложения, созданные пользователем.

DIMSTYLE - размерные стили, определенные в рисунке.

LTYPE - определения типов линий.

LAYER - характеристики слоев.

STYLE - описание гарнитур шрифтов.

VIEW - характеристики вида (сцены).

UCS - описания пользовательских систем координат.

VPORT - характеристики видовых экранов.

Каждый подраздел содержит различные элементы в количестве, достаточном для сохранения всей информации, присущей рисунку. Если в рисунке используются 10 слоев (включая слой 0), подраздел (или таблица) LAYER будет содержать 10 элементов.

programmer

DXF — стандартный формат файла обмена рисунками, с помощью которого Автокад обменивается информацией с внешним миром

APPID. Перечислены идентификаторы используемых программных приложений пользователя (или поставляемых с Автокадом).

DIMSTYLE. Содержит список размерных переменных и их значений для каждого поименованного размерного стиля.

LTYPE. Определены все типы линий, используемые в рисунке. Эти типы линий добавляются командой Автокада ТИПЛИН. К основным типам линий относятся DOT, DASHED, PHANTOM и DASHDOT. Тип линии CONTINUOUS по умолчанию всегда включается в DXF файл. Элементы подраздела содержат "образ" линии (в символах ASCII) в том виде, в каком она появится в рисунке.

LAYER. Содержит определение каждого слоя рисунка, т.е. ключевую информацию, необходимую при определении цвета, типа линии и видимости примитивов рисунка. Слои создаются по команде Автокада СЛОЙ/Новый. В Автокаде всегда по умолчанию присутствует слой с именем "0". Имя этого слоя должно восприниматься как символьная константа, а не как целое число.

STYLE. Содержит информацию о гарнитурах шрифтов и формах. В каждый элемент подраздела входят характеристики, задаваемые по умолчанию. В каждом элементе также записано имя файла шрифта или файла формы. Имя гарнитуры может быть таким же, как и имя файла. Даже если для одного и того же файла шрифта созданы несколько гарнитур, в подразделе каждая гарнитура описана отдельно. Автокад по умолчанию включает стандартную гарнитуру для файла TXT.SHX в DXF файл.

VIEW. Содержит список именованных видов и точек зрения, созданных в процессе рисования.

UCS. Перечислены именованные системы координат пользователя с точками отсчета и направлениями осей.

VPORT. Содержит список видовых экранов с координатами углов на экране дисплея, видимостью слоев в каждом из этих экранов и с указанием активных и текущего экранов на момент создания DXF файла.

BLOCKS

Данный раздел содержит детальную информацию о примитивах, входящих в каждый определенный в рисунке блок. Такие блоки могут быть созданы в результате нанесения размеров, штриховки, вставки других рисунков, библиотечных символов или внешних ссылок, а также в результате задания блоков в процессе рисования или в процессе работы прикладной программы. Даже стертые в рисунке и не видимые на экране дисплея блоки будут по-прежнему записаны в разделе BLOCKS DXF файла. При этом каждый блок будет представлен своим именем и точкой вставки, зарегистрированными в момент создания блока, за которыми следует список примитивов, составляющих блок. Каждый блок описывается в разделе индивидуально. Хотя и допустимо, чтобы один блок был составляющей частью другого блока, тем не менее описания блоков никогда не содержат вложенных описаний других блоков.

ENTITIES

В данном разделе перечислены все составляющие рисунок геометрические элементы, а также все детали, необходимые для воспроизведения рисунка. Отдельные примитивы могут ссылаться на информацию, записанную в предыдущих разделах (TABLES и BLOCKS). Описание каждого примитива имеет свою индивидуальную структуру.

Работа с DXF файлом

DXF файл построен по принципу признак-значение. Признак - это целое число, указывающее тип (целый, вещественный, строковый) и общее назначение (имя, координата, флаг вкл/откл) следующей за ним информации. Зная тип данных, который последует за признаком, можно переключиться на подпрограмму соответствующего преобразования данных. Все значения считываются как ASCII символы, и преобразование их к указанному в признаке представлению полностью возлагается на программиста. Признак и значение находятся на разных строках в DXF файле.

Признак записывается в формате I3 язы"а ФОРТРАН. Эт« «за́ачает, чт« »риза́а" - эт« це<«е чис<«, зааи»ающее три си»в«а, выраваёа́ых »« »рав«" граа́ице. На»ри»ер »риза́а", с«ст«ящи" из «да́«" цифры, будет и»еть два в»ереди ст«ящих »р«бе<а. Б«ьшиа́ств« »ри"<ада́ых »р«гра»» а́е а́уждается в с<ед«ваа́ии эт«у ф«р»ату, »«т«у чт« язы"и Pascal и C все рава́« «тбрасывают в»ереди ст«ящие »р«бе<ы. Са» Авт«"ад »ри чтеа́ии DXF фа"<«в та"же а́е а́астаивает а́а эт« ф«р»ате. Ода́а"« а́е"«т«рые другие »р«гра»»ы требуют и»еа́а« та"«г« ф«р»ата.

Авtокад следует очень строгой системе признаков типов данных. Это делает абсолютно несложным извлечение именно необходимой информации. Однако это означает также, что программа должна быть очень аккуратной при чтении и записи DXF файлов.

К примеру, давайте посмотрим на простейший файл со структурой признак-значение, похожей на DXF.

```
2
ЯБЛОКО
10
5.216
```

Файл содержит две порции информации: ЯБЛОКО и 5.216. Цифра 2, которая стоит перед словом "ЯБЛОКО", – это признак. В нашем примере 2 означает, что следом идет символьная строка (ЯБЛОКО). Цифра 10, которая стоит перед 5.216, - признак того, что в следующей строке стоит вещественное число (5.216).

Часть программы, которая распознает структуру такого файла, - простейший цикл. Управление остается в цикле WHILE до тех пор, пока не встретится конец файла. В заголовке цикла читается входной файл и извлекается из него признак. Однако случится большая беда, если в этот момент функция fscanf, прочтя строку из файла, не встретит ни признака, ни конца файла.

```
read_dxf   (dxf)
FILE       *dxf;
{
int        tag;
char       name[256];
char       garbage[256];
duble      real;
   while(fscanf(dxf,"%d",&tag) != EOF)
      switch (tag)
         {
            case   2  :   fgets(dxf,name,256);
                          break;
            case   10 :   fscanf(dxf,"%f",&real);
                          break;
```

```
            default         fgets(dxf,garbage,256);
                            break;
        }
}
```

Если отсутствует признак, мы потеряем управление и результат будет непредсказуемым. Вы можете добавить строки, которые будут проверять, является ли признак допустимым. Когда Автокад или другая программа генерирует DXFфайл, структуры будут допустимыми во всех случаях. Если же вручную создавать или исправлять DXF файл, тогда в него могут закрасться ошибки и неточности.

Теперь посмотрим, как работает эта небольшая программа. Читая признак и преобразуя его к целочисленному значению (в заголовке WHILE), определяем метод чтения и преобразования значения. Это реализовано в выражении switch. Для ввода строковых символов была выбрана функция fgets, которая помещает всю строчку файла в переменную, в то время как fscanf прекратит чтение, как только встретится первый пробел. Функцию fscanf допустимо использовать для чтения односложных имен (ЯБЛОКО). При чтении более сложной строки символов (ПО ЯБЛОКУ В ДЕНЬ) fscanf прочтет только первое слово в строке (ПО). Это сделает второе слово (ЯБЛОКУ) входным для следующего признака и изменит последовательность входного потока. Произойдет ошибка. Автокад с помощью признака дает знак, ожидать ли односложного имени или сложную строку символов.

Диапазон признаков:

0 - 9 - строки символов (имена, атрибуты, текст);

10 - 59 - вещественные числа (координаты, углы, расстояния);

60 - 79 - целые числа (переключатели, счетчики, наборы);

999 - строки комментариев;

1000 - 1009 - строки символов (имена, текст в расширенных данных);

1010 - 1059 - вещественные числа (координаты, углы, расстояния в расширенных данных);

1060 - 1079 - целые числа (в расширенных данных).

Признаки DXF файла.

0 - начало разделителя в DXF файле (начало раздела, таблицы или примитива). Строка символов без пробелов.

1 - основной текст. Встречается в примитивах TEXT, ATTRIB и ATTDEF. Строка символов, в которой могут быть пробелы.

2 - имя. В качестве имени блока - строка символов без пробелов. При использовании в качестве имени атрибута - строка символов, в которой могут быть пробелы.

3 - 4 - другие текстовые значения или имена. Строки символов, в которых могут быть пробелы.

5 - метки примитивов. Строка шестнадцатиричных символов без пробелов.

6 - имя типа линии. Строка символов без пробелов.

7 - имя гарнитуры шрифта. Строка символов без пробелов.

8 - имя слоя. Строка символов без пробелов.

9 - имя системной переменной в разделе HEADER. Строка символов без пробелов.

10 - основная координата X. Значением является строка ASCII символов, представляющая число с плавающей точкой.

11 - 18 - другие координаты X. Значением является строка ASCII символов, представляющая число с плавающей точкой.

20 - основная координата Y. Значением является строка ASCII символов, представляющая число с плавающей точкой.

21 - 28 - другие координаты Y. Значением является строка ASCII символов, представляющая число с плавающей точкой.

30 - основная координата Z. Значением является строка ASCII символов, представляющая число с плавающей точкой.

31 - 37 - другие координаты Z. Значением является строка ASCII символов, представляющая число с плавающей точкой.

38 - если не 0.0, - уровень объекта. Присутствует только в предыдущих версиях Автокада. Значением является строка ASCII символов, представляющая число с плавающей точкой.

39 - если не 0.0, - высота выдавливания объекта. Значением является строка ASCII символов, представляющая число с плавающей точкой.

40 - 48 - другие значения с плавающей точкой.

49 - повторяющееся значение. Используется для таблиц переменной длины (например, при описании штрихов типа линии). Значением является строка ASCII символов, представляющая число с плавающей точкой.

50 - 58 - углы (в градусах). Значением является строка ASCII символов, представляющая число с плавающей точкой.

62 - номер цвета. Значением является строка ASCII символов, представляющая целое число.

66 - признак "далее следует дополнительная последовательность информации". Используется при перечислении атрибутов в блоке или вершин в полилинии. Значением является строка ASCII символов, представляющая целое число.

70 - 78 - целое число, представляющее флаговые биты, режимы и т.д. Значением является строка ASCII символов, представляющая целое число.

210, 220, 230 - компоненты X,Y,Z направления выдавливания. Значением является строка ASCII символов, представляющая число с плавающей точкой.

999 - комментарии в DXF файле. Строка символов, в которой могут быть пробелы.

1000 - строка ASCII символов длиной до 255 байтов.

1001 - зарегистрированное имя приложения. Строка ASCII символов длиной до 31 байта.

1002 - управляющая строка расширенных данных примитива. Это либо "{" либо "}".

1003 - имя слоя в расширенных данных примитива.

1004 - двоичные данные, организованные в виде порций переменной длины не более 127 байтов.

1005 - метки примитивов.

1010, 1020, 1030 - вещественные числа, представляющие координаты.

1011, 1021, 1031 - вещественные числа, представляющие положение в мировом пространстве.

1012, 1022, 1032 - вещественные числа, представляющие смещение в мировом пространстве.

1013, 1023, 1033 - вещественные числа, представляющие направление в мировом пространстве.

1040 - вещественное число.

1041 - расстояние.

1042 - масштабный коэффициент.

1070 - 16-разрядное целое число (со знаком или без).

1071 - 32-разрядное (длинное) целое со знаком.

Признаки имеют специфическое использование и поэтому должны появляться в соответствующем контексте. Недопустимое значение признака приведет к тому, что Автокад натолкнется на ошибку, выдаст сообщение, прекратит чтение и завершит обработку DXF файла.

Итак, мы познакомились с идеей пар признак-значение, чтобы написать программу, умеющую интерпретировать данные из файла, подобного DXF. DXF файл полностью состоит из таких пар-примитивов. На следующем уровне сложности Автокад комбинирует ГРУППОВОЕ ИМЯ и соподчиненные ему значения. Каждое ГРУППОВОЕ ИМЯ и связанное с ним значение тоже пара признак-значение. Например, примитив LINE состоит как минимум из пяти пар признак-значение в группе line:

```
0
LINE
10
1.0
20
1.5
11
```

6.32
21
7.223

Здесь показана очень простая группа line, поскольку типичная группа line может содержать информацию о слое, цвете, высоте и уровне. Признак 0 предупреждает о начале примитива. Он также отмечает начало блоков, разделов, таблиц, но здесь нас это не интересует. Значение, следующее за признаком 0, говорит о том, какой примитив представлен в оставшейся части группы. Распознав значение "LINE", прикладная программа вызовет функцию-обработчик примитива LINE.

Приведенный в примере отрезок (примитив LINE) расположен между точкой с координатами 1.0, 1.5 и точкой с координатами 6.32, 7.233. За описанием группы следуют пары признак-значение, содержащие в себе эти координаты.

Чтение DXF файла

Autodesk спроектировал DXF файл так, что один центральный модуль вызывает анализаторы отдельных разделов. Анализатор - это модуль, который "разбирает на части" структуру раздела так, чтобы части информации можно было использовать независимо.

Анализатор выполняет операцию, напоминающую синтаксический разбор предложения на подлежащее-сказуемое-обороты. Разбирать DXF файл гораздо проще, чем разбирать предложения естественного языка. В DXF файле каждый раздел начинается с признака раздела и значения SECTION:

0
SECTION

Когда программа видит эту пару, она должна начать определять тип раздела. Информация об этом располагается непосредственно вслед за строкой SECTION. Тип раздела - это пара с признаком 2 и именем раздела. Например:

2
HEADER

Как только программа распознала имя раздела, следующее за признаком 2, она вызывает соответствующий анализатор раздела. В данном примере будет вызван анализатор раздела HEADER. Структура информации, расположенной ниже имени раздела, имеет значение только для конкретного анализатора.

Формат DXF файла определяет также конец раздела. По существу конец раздела означает, что управление должно передаться в центральный анализирующий модуль. Тем самым центральный модуль подготавливается к распознаванию следующего раздела и вызову очередного анализатора (анализатора заголовка, таблиц, блоков или примитивов). Конец раздела тоже пара признак-значение с признаком 0 и значением ENDSEC:

0
ENDSEC

Конец DXF файла отмечается парой
0
EOF

что позволяет центральному модулю сделать вывод, что чтение DXF файла завершено.

Мы надеемся, что читатель получил некоторое представление о структуре файла обмена геометрической информацией Автокада, методах его чтения и разбора. Желаем успеха в увлекательном деле программирования Автокада. В будущих статьях мы расскажем о других подходах к этой проблеме.

В прилагаемом к статье листинге представлены некоторые основные структуры данных DXF файла. Они могут помочь в проектировании собственных прикладных программ на языке программирования С, обрабатывающих или генерирующих DXF файлы.

(Примечания. Примитивы TEXT, ATTDEF и ATTRIB используют структуру "text_type". Примитивы TRACE и SOLID - "type_trace".)

Листинг

```
typedef
    struct  dash_type
        {
            struct  dush_type   *next;
            double              length;
        }                   dush_type;
typedef
    struct  ltype_type
        {
            struct  ltype_type  *next;
            char                *name;
            char                *descript;
            int                 flags;
            int                 alignment;
            int                 dashes;
            double              length;
            struct  dash_type   *dash_length;
        }                       ltype_type;
Определение структуры данных типа линии.
typedef
    struct  layer_type
        {
            struct  layer_type  *next;
            char                *name;
            char                *descript;
            int                 flags;
            int                 color;
            int                 used;
            char                *lt;
        }                       layer_type;
Определение структуры данных слоя.
typedef
    struct  style_type
        {
            struct  style_type  *next;
            char                *name;
            int                 flags;
            int                 gen_flags;
            double              txt_scale;
            double              height;
            double              oblique;
            double              last_height;
            char                *font;
            char                *big_font;
        }                       style_type;
Определение структуры данных гарнитуры
typedef
    struct  view_type
        {
            struct  view_type   *next;
            char                *name;
            int                 flags;
            double              v_height;
            double              v_width;
            double              v_ctr_x;
            double              v_ctr_y;
            double              v_dir_x;
            double              v_dir_y;
            double              v_dir_z;
            double              o_locate_x;
            double              o_locate_y;
            double              o_locate_z;
            double              lens;
            double              front_clip;
            double              back_clip;
            double              persp_height;
            double              angle;
            int                 v_mode
        }                       view_type;
Определение структуры данных вида.
typedef
    struct  ucs_type
        {
            struct  ucs_type    *next;
            char                *name;
            int                 flags;
            double              origin_x;
            double              origin_y;
            double              origin_z;
            double              x_dir_x;
            double              x_dir_y;
            double              x_dir_z;
            double              y_dir_x;
            double              y_dir_y;
            double              y_dir_z;
```

```
    }                       ucs_type;
```
Определение структуры данных ПСК.

```
typedef
    struct  block_type
        {
            struct  block_type  *next;
            int                 what;
            char                *name;
            char                *xref_name;
            int                 flags;
            layer_type          *lyr;
            ltype_type          *ltype;
            double              base_x;
            double              base_y;
            double              base_z;
        }                       block_type;
```
Определение структуры данных описания блока.
```
typedef
    struct  line_type
        {
            struct  line_type   *next;
            int                 what;
            char                *lyr;
            char                *lt;
            double              sx;
            double              sy;
            double              sz;
            double              ex;
            double              ey;
            double              ez;
            double              w1;
            double              w2;
            double              bulge;
            char                last_one;
            struct  line_type   *vertex;
        }                       line_type;
```
Определение структуры данных линии (отрезка, сегмента полилинии).

```
typedef
    struct point_type
        {
            struct  point_type  *next;
            int                 what;
            char                *lyr;
            char                *lt;
            double              px;
            double              py;
            double              pz;
            double              x_ang;
        }                       point_type;
```
Определение структуры данных точки.

```
typedef
    struct circle_type
        {
            struct  circle_type *next;
            int                 what;
            char                *lyr;
            char                *lt;
            double              cx;
            double              cy;
            double              cz;
            double              r;
        }                       circle_type;
```
Определение структуры данных круга.

```
typedef
    structarc_type
        {
            struct  arc_type    *next;
            int                 what;
            char                *lyr;
            char                *lt;
            int                 flags;
            double              acx;
            double              acy;
            double              acz;
double ar;
double sa;
double ea;
double width;
} arc_type;
```
Определение структуры данных дуги.

```
typedef
    struct trace_type
        {
            struct trace_type *next;
            int what;
            char *lyr;
            char *lt;
            double x1;
            double y1;
            double z1;
            double x2;
            double y2;
            double z2;
            double x3;
            double y3;
            double z3;
            double x4;
            double y4;
            double z4;
        } trace_type;
```
Определение структуры данных полосы и фигуры.
```
typedef
struct text_type
{
struct text_type *next;
int what;
char *lyr;
char *lt;
double ix;
double iy;
double iz;
double height;
char *text;
char *prompt;
char *def_value;
char *tag;
int field_length;
int att_flags;
double angle;
double x_scale;
double oblique;
style_type *font;
int gen_flags;
int x_justify;
int y_justify;
double ax;
double ay;
double az;
} text_type;
```
Определение структуры данных текста.
```
typedef
struct shape_type
{
struct shape_type *next;
int what;
char *lyr;
char *lt;
double ix;
double iy;
double iz;
double size;
char *shape_name;
double angle;
double x_scale;
double oblique;
} shape_type;
```
Определение структуры данных формы.
```
typedef
struct insert_type
{
struct insert_type *next;
int what;
char *lyr;
char *lt;
int att_flags;
char *b;
double ix, iy, iz;
double angle;
double x_scale, y_scale z_scale;
int rows;
int columns;
double r_space;
double c_space;
} insert_type;
```
Определение структуры данных вставки блока.

Начиная с этого номера, журнал "Монитор" будет публиковать информацию о новых вирусах в формате антивирусной программы AVSP 2.0. Большинство сообщений о вирусах будет содержать информацию, достаточную как для обнаружения, так и для лечения вирусов.

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов.

AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженых новых вирусах.

Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса: Jews-2339
Номер: 77.1-37197
Тип вируса: PROGRAM
Размер: 2339
Шаблон: 008C8448008B844200A30600EB
Процедуры: *.SYS { Opt(St:27) Mov(End:-5,Int:0,5) Mov(St:66,Head:6,2) Tr(St:0) } *.COM { Opt(End:-2312) Mov(End:-5,Beg:0,3) Tr(End:-2339) }
Справка: Содержит строку "Jews-2 Virus. MSU 1991". Заражает драйверы (SYS файлы).

Имя вируса: Jews-2370
Номер: 78.1-10794
Тип вируса: PROGRAM
Размер: 2370
Шаблон: 81EE6701E8F30256BF000181C63D09
Процедуры: *.SYS { Opt(St:360) Mov(End:-5,Int:0,5) Mov(St:68,Head:6,2) Tr(St:0) } *.COM { Opt(End:-2010) Mov(End:-5,Beg:0,3) Tr(End:-2370) }
Справка: Содержит строку "Jews-2 Virus. MSU 1991". Заражает драйверы (SYS файлы).

Имя вируса: Hello
Номер: 204.1-3553
Тип вируса: PROGRAM
Размер: 751
Шаблон: B44ECD21731AE934018B1ECE02
Процедуры: *.COM { Opt(End:-700) Mov(MP:400,Beg:0,3) Tr(MP:-51) }
Справка: Содержит строку "Hello! Y have got a virus for you!"

Имя вируса: V-460
Номер: 220.1-39241
Тип вируса: PROGRAM
Размер: 460
Шаблон: 813E6401D32E7449C7066401D32E
Процедуры: *.EXE { Opt(St:14) Mov(MP:442,H:20,2) Mov(MP:440,H:22,2) TrEXE(St:0) }

Имя вируса: V-167
Номер: 209.1-40235
Тип вируса: PROGRAM
Размер: 167
Шаблон: 5E0374FEFF74FCFF74FE50
Процедуры *.COM { Opt(End:-163) Mov(MP:-4,Beg:0,4) Tr(MP:-4) }
Справка: Только размножается.

По вопросам приобретения AVSP 2.0 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно.

Цена одного комплекта AVSP 2.0 на 3 сентября 1992 г. составляет 1924 рубля (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров - до 60%.

MERISEL : опыт цивилизованной работы с дилерами

Еще совсем недавно закупки вычислительной техники и программного обеспечения осуществлялись, как правило, непосредственно у фирм-производителей. С появлением фирм - дистрибьютеров, которые за счет больших объемов поставок получают от производителей крупные скидки и представляют сразу несколько производителей, стало выгодным приобретать товар именно у них. Однако сами дистрибьютеры не занимаются розничной торговлей. Связующим звеном между дистрибьютером и покупателем является дилер.

Компания "Merisel-Компьютерные технологии" появилась на рынке вычислительной техники и программных средств в 1990 году. Учредитель компании - торгово-оптовая корпорация Merisel, один из лидеров американского рынка. "Merisel-Компьютерные технологии" начала свою деятельность с активного формирования дилерской сети. Сегодня, спустя 2 года, дилерская сеть компании включает более 250 дилеров и охватывает почти все государства СНГ и Балтии. Это позволило компании стать крупнейшим в СНГ дистрибьютером программных продуктов и меньше чем за год перейти от розничной торговли к оптовым поставкам через дилерскую сеть. У компании Merisel Вы можете приобрести самые популярные программные продукты фирм - лидеров компьютерной индустрии: Word Perfect, Lotus Development, Borland, Symantec, Corel Systems, Aldus. Специалистами компании созданы русские версии систем Norton Commander, Norton AntiVirus, Norton Utilities, Norton Backup фирмы Symantec, в ближайшее время будет закончена работа по локализации CorelDraw 3.0 фирмы Corel Systems и AmiPro фирмы Lotus. Merisel предлагает русские версии и других программных продуктов. В результате деятельности Merisel существенно удлинился список высококачественных программных продуктов, доступных для наших покупателей за рубли. Компания также осуществляет поставки вычислительной техники высокого качества производства таких фирм, как Intel, Hewlett-Packard, AST Research, Seiko Epson, Seagate.

Merisel уделяет очень большое внимание отбору и обучению дилеров. Ставшая популярной не только в США, но и в Европе, Канаде, Австралии и Латинской Америке программа обучения дилеров Softeach реализуется теперь и в России. Softeach - это специальное обучение дилеров, которое проводят представители фирм-производителей вычислительной техники и программных средств. Программа рассчитана в первую очередь не на пользователя, а на продавца. В ходе семинаров дилеры знакомятся с новыми продуктами, получают "из первых рук" информацию об их особенностях и рекомендации по эффективной продаже. В ходе Softeach Вы можете получить ответы на все интересующие Вас вопросы от представителей фирм, получить пакеты рекламно-информационных материалов по представленным продуктам, заключить на льготных условиях контракт, принять участие в лотереях и конкурсах. Фирмы-производители с удовольствием принимают участие в этой программе, так как они имеют возможность получить информацию о продажах своей продукции в конкретном регионе и обсудить с дилерами связанные с этим проблемы. Первый форум дилеров Softeach прошел в апреле 1992 года в Москве в период работы выставки COMTEK'92. В серии семинаров-презентаций приняли участие представители как хорошо известных на нашем рынке фирм: Borland, Corel Systems, Intel, Lotus Development, Logitech, Microsoft, Symantec, Word Perfect, так и мало знакомых: APC, Cyrix, Dynamic Pathways Company. По оценке большинства дилеров, участвовавших в Softeach (а всего их было более 700 человек), объем информации, полученной в ходе форума, значительно превышает возможности любой выставки. Следующий Softeach будет приурочен к выставке COMTEK'93.

В течение всего 1993 года будет реализовываться еще одна программа поддержки дилеров - Frequent Buyer ("Постоянный покупатель"). Цель программы - поощрение наиболее успешно работающих дилеров. Участники программы Frequent Buyer будут получать призовые очки как за объем продаж в целом, так и за продажу отдельных продуктов. Победители получат в качестве призов программные продукты и товары народного потребления либо смогут использовать набранные призовые очки для оплаты маркетинговых мероприятий, рекламы, обучения за границей, изготовления сувениров, совместно с компанией "Merisel-Компьютерные технологии" участвовать в выставках.

Готовится к реализации также программа абонементного информационного обслуживания дилеров. Каталог фирмы Merisel насчитывает почти 10 тысяч наименований вычислительной техники и программных средств более чем 700 производителей. По запросам дилеров-абонентов компания будет поставлять информацию по отдельным продуктам или информацию по продукции отдельных фирм. Кроме того, абоненты будут периодически получать информацию по всем новым продуктам, появляющимся в каталоге.

Формирование дилерской сети компании "Merisel-Компьютерные технологии" продолжается.

Подробную информацию об условиях сотрудничества с компанией Вы можете получить по телефонам (095)276-90-08, (095)276-47-18. Факс (095)274-00-97.

Лучшая продукция и компетентность дилеров—вот что поднимает MERISEL к высотам коммерческого успеха.

Если Вы дилер высокого класса, сотрудничайте с фирмой MERISEL и тогда MERISEL предложит Вам:

- льготную систему вознаграждения
- широкую программу обучения
- содержательные семинары, форумы и выставки.

Если Вы дилер MERISEL , Вы предложите своим клиентам:

- 10 000 наименований высококачественной техники 700 ведущих фирм мира
- цены на 15—20% ниже европейских

и еще:

- комплектование индивидуальных заказов и поставку оборудования со складов в Москве
- техническое обслуживание вычислительной техники
- информационно—консультативное сопровождение программных средств
- регистрацию пользователей программных средств и поставку новых версий.

Высокий бизнес ждет Вас, господа дилеры,

а Merisel ждет Вас по адресу:

109044 Москва, Крутицкий вал, д.3, корп. 2.
Телефоны: (095) 276—9008, (095) 276—3149, (095) 276—4718.
Факс: (095) 274—0097.

MERISEL C.A.T.Ltd

Софт·Москва

РАЗРАБОТКА ЭЛЕМЕНТОВ ФИРМЕННОГО СТИЛЯ

Издательские Услуги:

БРОШЮРЫ

ЛИСТОВКИ

КАТАЛОГИ

ДОКУМЕНТАЦИЯ

И ДРУГАЯ ПЕЧАТНАЯ ПРОДУКЦИЯ

Тел. (095) 247—36—25,
(095) 903—46—26 (после 19°°).
Факс. (095) 126—31—56.